REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 3, 1998	3. REPORT TYPE AND DATES COVERED Final report

4. TITLE AND SUBTITLE
Parallel-Computing Concepts and Methods in Large Scale Floquet Analysis of Helicopter Trim and Stability

5. FUNDING NUMBERS
DAAH 04-94-G-0185

6. AUTHOR(S)
G. H. Gaonkar, S. Subramanian and S. Venkataratnam

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)
Florida Atlantic University
Department of Mechanical Engineering
777 Glades Road
Boca Raton, FL 33431

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
ARO 32174.15-EG

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

19980519 098

13. ABSTRACT *(Maximum 200 words)*

Rotorcraft stability investigation involves a nonlinear trim analysis for the control inputs and periodic responses, and, as a follow-up, a linearized stability analysis for the Floquet transition matrix (FTM), and its eigenvalues and eigenvectors. The trim analysis is based on a shooting method with damped Newton iteration, which gives the FTM as a byproduct, and the eigenanalysis on the QR method; the corresponding trim and stability analyses are collectively referred to as the Floquet analysis. A rotor with Q blades that are identical and equally spaced has Q planes of symmetry. Exploiting this symmetry, the fast-Floquet analysis, in principle, reduces the run time and frequency indeterminacy of the conventional Floquet analysis by a factor of Q. It is implemented on serial computers and on all three types of mainstream parallel-computing hardware: SIMD and MIMD computers, and a distributed computing system of networked workstations; large models with hundreds of states are treated. A comprehensive database is presented on computational reliability such as the eigenvalue condition number and on parallel performance such as the speedup and efficiency, which show, respectively, how fast a job can be completed with a set of processors and how well their idle times are minimized. Despite the Q-fold reduction, the serial run time is excessive and grows between quadratically and cubically with the number of states. By contrast, the parallel run time can be reduced dramatically and its growth can be controlled by a judicious combination of speedup and efficiency. Moreover, the parallel implementation on a distributed computing system is as routine as the serial implementation on a workstation.

14 SUBJECT TERMS
Helicopter Trim and Stability, Fast-Floquet Analysis, Massively Parallel and Distributed Computing

15. NUMBER IF PAGES
59

16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

Summary

Rotorcraft stability investigation involves a nonlinear trim analysis for the control inputs and periodic responses, and, as a follow-up, a linearized stability analysis for the Floquet transition matrix (FTM), and its eigenvalues and eigenvectors. The trim analysis is based on a shooting method with damped Newton iteration, which gives the FTM as a byproduct, and the eigenanalysis on the QR method; the corresponding trim and stability analyses are collectively referred to as the Floquet analysis. A rotor with Q blades that are identical and equally spaced has Q planes of symmetry. Exploiting this symmetry, the fast-Floquet analysis, in principle, reduces the run time and frequency indeterminacy of the conventional Floquet analysis by a factor of Q. It is implemented on serial computers and on all three types of mainstream parallel-computing hardware: SIMD and MIMD computers, and a distributed computing system of networked workstations; large models with hundreds of states are treated. A comprehensive database is presented on computational reliability such as the eigenvalue condition number and on parallel performance such as the speedup and efficiency, which show, respectively, how fast a job can be completed with a set of processors and how well their idle times are minimized. Despite the Q-fold reduction, the serial run time is excessive and grows between quadratically and cubically with the number of states. By contrast, the parallel run time can be reduced dramatically and its growth can be controlled by a judicious combination of speedup and efficiency. Moreover, the Floquet analysis of large models on a distributed computing system is as routine as the current treatment of relatively small-order models, not many more than 100 states, on a workstation.

Nomenclature

Unless otherwise stated, the symbols below are nondimensional:

a	lift curve slope, rad^{-1}
C_{d_0}	constant profile drag coefficient
C_d	resultant profile drag force in the plane of the rotor disk opposite to the flight direction
C_l	rolling moment coefficient
C_m	pitching moment coefficient
C_T	thrust coefficient
C_w	weight coefficient of the helicopter
$\overline{f}$	equivalent flat plate area of parasite drag
f	sequential fraction
M	number of states or state variables augmented with control inputs, $M = N + c$
N	number of structural and aerodynamic states or state variables
N_b	number of blade states
N_w	number of dynamic wake states
p	number of processors
P_β	flap natural frequency, rotating
Q	number of blades
t	time unit such that $T = 2\pi$
T	period
z_k	k-th eigenvalue of the FTM; see Eq. (2)
$\overline{z}_k$	k-th eigenvalue of EFTM; see Eq. (14)
α_j^r, β_j^r	wake states
ψ	azimuthal position
ξ_k	k-th mode nonunique frequency of z_k
$\overline{\xi}_k$	k-th mode nonunique frequency of $\overline{z}_k$
ΔT	T/Q
ϵ	perturbation quantity
γ	Lock number (blade inertia parameter)

σ	rotor solidity
σ_k	k-th mode damping of z_k
$\overline{\sigma}_k$	k-th mode damping of $\overline{z}_k$
μ	advance ratio
ω_ζ	lag natural frequency, rotating
$[\,]^T$	transpose of $[\,]$
$\|\ \|$	Euclidean norm of a vector or matrix
$(\dot{\mathbf{x}})$	time derivative of $\mathbf{x}$

List of Figures

List of Tables

1 Introduction

Floquet theory provides a mathematical basis to investigate rotorcraft stability in trimmed flight. The investigation requires a nonlinear trim analysis for the control inputs and the corresponding periodic responses, and then a stability analysis linearized about the periodic responses for the modal frequencies and damping levels. The periodic shooting method is widely used for the trim analysis since it is not sensitive to the damping levels or stability margins, and in combination with damped Newton-type iteration it is almost globally convergent; the Floquet transition matrix (FTM) comes out as a byproduct as well. The stability analysis then boils down to the eigenanalysis of the FTM, for which the QR method is used almost exclusively. In this report, the trim analysis with the shooting method and the stability analysis with the QR method are collectively referred to as the Floquet analysis. The bulk of the applications of the Floquet analysis is based on "classical" Floquet theory and serial computing; for the state of the art through 1987, see Gaonkar and Peters (1987). For the developments since, we mention, as examples, Achar and Gaonkar (1993) for the trim analysis, Achar and Gaonkar (1994), and Nagabhushanam and Gaonkar (1995) for the stability analysis, and Ravichandran et al., (1990) for the computational aspects. These studies show that the Floquet analysis can be applied confidently with a built-in means of quantifying the computational reliability. However, the treatments are restricted generally to relatively small-order models, not many more than 100 states. This restriction, and its reason and ramification provide the setting for this study and merit some elaboration.

The computer run time required for Floquet analysis grows rapidly with the number of states or order. As we strive to improve predictions by exploiting major strides in modeling the helicopter structure and its flow field, the run time becomes prohibitive. Such a run-time requirement permits neither the use of the state-of-the-art models nor the application of comprehensive design analyses that involve large models with hundreds of states ($N >> 100$). For example, it is not practical to generate the FTM by a full finite-element method with the state-of-the-art beam-element models. Thus, the utility of the Floquet analysis becomes severely limited. Significantly, the trim analysis consumes nearly 99% of the run time because it is a computer-intensive exercise that couples the nonlinear differential equations of motion with the algebraic-transcendental equations of trim. It involves predicting the control inputs that satisfy the flight conditions and then predicting (given those controls) the initial conditions that guarantee periodic responses. These control inputs appear not only in the damping and stiffness matrices but in the input matrix as well and are specified indirectly so as to satisfy flight conditions such as prescribed thrust level and force-moment equilibrium. Therefore, the trim analysis requires an iterative strategy of solving a sequence of linearized problems: varying or perturbing the starting or initial values of the state variables and control inputs *one at a time*, integrating *through one complete period* T and then improving the estimates by damped Newton iteration; the cycle of perturbing, integrating and improving continues till convergence. In short, the procedure requires a formidable number of operations of perturbation, integration and improvement; specifically, for a rotorcraft model with N states, c control inputs and k iteration cycles for convergence, $k(N+c+1)$ operations are required, out of which $(N+c+1)$ operations in each cycle are independent. It turns out that for a Q-bladed rotor, the time interval for all of the $k(N+c+1)$ integration operations can be reduced from T to T/Q by exploiting the Q-fold symmetry of the rotor; the only condition is that all the Q blades be identical and equally spaced, as is the case in practice anyway. The conventional Floquet analysis that exploits this symmetry becomes the fast-Floquet analysis. The benefits of this exploitation are significant: the run time and frequency indeterminacy of the conventional analysis are both reduced by a factor of Q (McVicar and Bradley, 1995; Peters, 1995; Chunduru, 1995; Chunduru et al., 1997; Subramanian, 1996; Subramanian and Gaonkar, 1996; Venkataratnam et al., 1997). Although the run time is reduced, it still grows rapidly between quadratically and cubically with the order, and the barrier of run-time constraint still remains for large models (Chunduru, 1995; Subramanian, 1996). It also turns out that the $(N+c+1)$ operations of perturbation, integration and improvement can be done concurrently by exploiting parallel computing. Indeed, parallel computing reduces the run time dramatically, theoretically by a factor of $(N+c+1)$. Equally significant, it also provides a means of controlling the growth of run time with the order by judiciously increasing the number of processors with the order as a compromise between reducing the run time and avoiding processor underutilization (Subramanian et al., 1996). The parallel fast-Floquet analysis exploits both the fast-Floquet analysis and parallel computing and thereby reduces the run time by a factor as large as $Q(N+c+1)$.

The parallel fast-Floquet analysis is implemented on massively parallel SIMD-MasPar MP1 (Single Instruction, Multiple Data) and MIMD-IBM SP2 (Multiple Instruction, Multiple Data) computers as well as on a distributed computing system of networked workstations, and models with hundreds of states are treated. A comprehensive database is presented, which includes computational reliability results such as a condition number of the Jacobian matrix in Newton iteration, and parallel-performance metrics such as the serial and parallel run times and their rates of growth with the order. Among these metrics, efficiency merits special mention since it shows how effectively the processors are used to speed up the computations, allowing the analyst to guard against processor underutilization. Although the Floquet analysis is tested on all three types of mainstream parallel-computing hardware, the distributed computing system is emphasized throughout. As far as a user's program is concerned, distributed computing system acts as one single virtual parallel computer, no matter how many workstations are connected, how different they are and where they are located. It provides high-performance computing with minimal cost and turnaround time, and makes parallel computing practical; aptly referred to as "lowly parallel computing" (Pfister, 1995), it is no more involved than implementing a serial program on a workstation (Venkataratnam et al., 1997). In summary, the parallel fast-Floquet analysis removes the barrier of run-time constraint that now prevents the routine application of Floquet theory to large models. This is a measure of its utility.

2 Parallel Computing Hardware

Sequential algorithms are essentially interchangeable, say from DEC VAXs to IBM RS6000s to workstations. Such a portability does not yet exist for parallel algorithms, which are affected by two types of architecture: SIMD and MIMD. For massively parallel computing per se, the MIMD computers are dominating the field, at least with respect to cost and speed; they utilize the "off-the-shelf" processors, the same used in workstations, and exploit a chip technology that is making exponential growth in processing speed. In fact, by the turn of the century the processing rate is expected to exceed 200 MFLOPS (Snir et al., 1996). The algorithms developed for a distributed computing system can be directly implemented on a MIMD computer. However, an implementation on a SIMD computer would require modifications (Venkataratnam et al., 1997). The two architectures and their impact on the development of the algorithms are extensively discussed in the literature (e.g., Kumar et al., 1994). Although from an algorithmic perspective, the distributed computing system belongs to the MIMD architecture, it is a relatively recent development, mostly since the early '90s. A brief account of it is included in the sequel, primarily to provide a better appreciation for the simplicity and utility of the parallel fast-Floquet analysis based on distributed computing. This account is intentionally descriptive with minimal use of parallel-computing jargon; for a thorough account see Snir et al., (1996).

2.1 Distributed Computing

Distributed computing or heterogeneous network computing involves a set of computers connected by a network that can range from a general-purpose workstation to a high-performance computer to even a parallel computer. In practice, however, a distributed computing system represents an assemblage of heterogeneous workstations networked together, and its computational power can be comparable to that of a super computer. This networking in no way interferes with the stand-alone operation of individual workstations; indeed, one of the drivers of distributed computing is the necessity to combine and realize the unutilized computational power of these individual workstations during off-load (e.g., after-office) hours. This unutilized computing power merits special consideration since a typical workstation routinely delivers several tens of millions of floating point operations per second (MFLOPS) and its computing power is expected to increase rapidly. Computing facilities from academia to design offices provide access to a large number of workstations. The bulk of these workstations sits idle for most of the day. Distributed computing provides a means of harnessing this untapped computing power.

In contrast, a massively parallel computer represents an assemblage of a few hundred to a few thousand identical computers or processors and provides enormous computational power. Despite the architectural contrast — heterogeneous versus identical processors — both the distributed computing systems and massively parallel computers use the same programming paradigm for interprocessor communication, which ensures all processors understand and exchange data. Although several software programs for interprocessor communication have been proposed by different vendors, the MPI (Message Passing Interface) in the public domain has evolved as a standard. The user need not worry about the architectural differences among the processors and related consequences such as distribution of tasks and data exchange. In other words, the MPI library spans the collection of heterogeneous processors in a distributed computing system as it does in a massively parallel computer with identical processors, although in the latter case this spanning is done much faster.

The increasing adaptation of distributed computing systems with massively parallel computers as necessary or highly desirable adjuncts is due to two factors. First, distributed computing involves very little cost since it is built up on existing hardware. By comparison, state-of-the-art massively parallel computers typically cost more than $10 million, and consequently they are maintained by only a few organizations. Since they are heavily used, the turnaround time often runs into days for a typical large-scale Floquet analysis ($N \approx 400$). Second, common to both types is the message passing programming paradigm (e.g., MPI); this means the same algorithm or code developed in a distributed computing system can be run on a massively parallel computer as well. Thus, in the development of a prediction code, the bulk of the computations can be done on a distributed computing system at very little cost and turnaround time, and the 'final-stage' computations and other demonstration models can be run on a massively parallel computer. Stated otherwise, a judicious combination of distributed and massively parallel computing provides a practical means of addressing the current run-time constraint that now prevents the routine application of Floquet theory to large models.

3 Conventional- and Fast-Floquet Analyses

A Q-bladed rotor with identical and equally spaced blades has Q planes of symmetry. This means the fundamental solution matrix or the transition matrix over one complete period T can be constructed from the transition matrix over T/Q or ΔT. The fast-Floquet analysis exploits this symmetry in the trim analysis as well as in the stability analysis. The exploitation primarily translates into describing the trim equations in a Q-th part of a revolution and then *integrating* the equations of motion through ΔT, not through T as in the conventional analysis. As a byproduct, it also translates into generating an equivalent Floquet transition matrix (EFTM), whose eigenvalues not only lead to the stability results directly but they also reduce the frequency indeterminacy by a factor of Q. Thus, we obtain the Floquet analysis in $1/Q$ of the run time of the conventional analysis. However, the fast-Floquet analysis calls for substantial changes in the conventional analysis to properly account for the Q-fold symmetry in the trim and stability analyses, and these changes have considerable bearing on the subsequent parallelization. To help explain these changes and related benefits of run-time saving and mode identification, we begin with the stability and response of a linear system and then outline the conventional shooting strategy. This is followed by a discussion of the fast-Floquet theory for one-rotored and multirotored models.

3.1 Conventional Floquet Analysis

For a linear periodic-coefficient system with $N \times 1$ state vector $\mathbf{x}(t)$, the equations of motion can be written as

$$\dot{\mathbf{x}}(t) = \mathbf{A}(t)\,\mathbf{x}(t) + \mathbf{G}\,(t) \tag{1}$$

where $\mathbf{A}(t)$ and $\mathbf{G}(t)$ represent the state matrix and forcing function or input vector with period $T = 2\pi$, and are of size $N \times N$ and $N \times 1$, respectively. The $N \times N$ state transition matrix $\phi(t)$ is the fundamental solution of the matrix differential equation:

$$\dot{\phi} = \mathbf{A}(t)\phi, \quad 0 \le t \le 2\pi, \quad \phi(0) = \mathbf{I} \tag{2}$$

By definition, $\phi(2\pi)$ is the FTM, and its eigenvalues, z_k, determine system stability. The modal damping levels and frequencies are computed from z_k:

$$\sigma_k = \frac{1}{2\pi} \ln |z_k| \tag{3a}$$

$$\xi_k = \frac{1}{2\pi} \arg(z_k) = \frac{1}{2\pi} \tan^{-1} \left(\frac{\mathrm{Im}(z_k)}{\mathrm{Re}(z_k)} \right) \tag{3b}$$

For the complete or nonhomogeneous Eq. (1), the initial conditions that guarantee periodic forced response, that is, $\mathbf{x}(0) = \mathbf{x}(2\pi)$, are given by

$$[\mathbf{I} - \phi(2\pi)]\,(\mathbf{x}(0) - \mathbf{x}_E(0)) = (\mathbf{x}_E(2\pi) - \mathbf{x}_E(0)) \tag{4}$$

where $\mathbf{x}_E(2\pi)$ is the non-periodic solution at $t = 2\pi$ for any arbitrary initial state $\mathbf{x}_E(0)$.

In general, rotorcraft systems are governed by nonlinear periodic-coefficient equations, which can be represented as

$$\dot{\mathbf{x}} = \mathbf{G}\,(\mathbf{x}, t) \tag{5}$$

For such a nonlinear system, the initial conditions to generate periodic response can be obtained by applying a Newton-type iteration. Specifically, for the k-th iteration, Eq. (4) reads:

$$\mathbf{x}(0)_{k+1} = \mathbf{x}_E(0)_k + \chi\,[\mathbf{I} - \phi(2\pi)]_k^{-1}\,(\mathbf{x}_E(2\pi) - \mathbf{x}_E(0))_k \tag{6}$$

where $\phi(2\pi)$ converges to the FTM. The matrix $[\mathbf{I} - \phi(2\pi)]$ is the Jacobian or partial derivative matrix Φ, and χ is the Newton damping parameter.

With the control-input vector $\mathbf{c}$ shown explicitly, the rotorcraft equations of motion can be expressed as

$$\dot{\mathbf{x}} = \mathbf{G}(\mathbf{x}, \mathbf{c}, t) \tag{7}$$

We solve the above equation for the initial state $\mathbf{x}(0)$ that gives periodic response as well as for the control-input vector $\mathbf{c}$ that gives desired flight conditions of force-moment balance. Explicitly stated, the initial state $\mathbf{x}(0)$ and contorl input $\mathbf{c}$ satisfy:

$$\mathbf{x}\,(2\pi, \mathbf{x}(0)) - \mathbf{x}(0) = \mathbf{0} \tag{8}$$

$$\mathbf{f}(\mathbf{x}, \mathbf{c}) = \mathbf{0} \tag{9}$$

where $\mathbf{x}(2\pi, \mathbf{x}(0))$ is the state at $t = 2\pi$ with initial state $\mathbf{x}(0)$, and $\mathbf{f}(\mathbf{x}, \mathbf{c})$ symbolically represents the force-moment balance. Equations (8) and (9) are nonlinear algebraic-transcendental equations, which, when combined, can be written as

$$\mathbf{f}(\mathbf{s}) = \mathbf{0} \tag{10}$$

where $\mathbf{s} = \lfloor \mathbf{x},\ \mathbf{c} \rfloor^T$ represents the augmented-state vector of state variables and control inputs.

Equation (6) extended to account for Eq. (9) can be expressed as (Achar and Gaonkar, 1993):

$$\left\{ \begin{array}{c} \mathbf{x}(0) \\ \mathbf{c} \end{array} \right\}_{k+1} = \left\{ \begin{array}{c} \mathbf{x}_E(0) \\ \mathbf{c} \end{array} \right\}_{k} - \chi \left[\begin{array}{cc} \mathbf{\Phi}_{11} - \mathbf{I} & \mathbf{\Phi}_{12} \\ \mathbf{\Phi}_{21} & \mathbf{\Phi}_{22} \end{array} \right]^{-1} \left\{ \begin{array}{c} \mathbf{x}_E(2\pi) - \mathbf{x}_E(0) \\ \boldsymbol{\delta} \end{array} \right\} \tag{11}$$

where $\boldsymbol{\delta}$ is the error in satisfying Eq. (9); that is, $\mathbf{f}(\mathbf{x}, \mathbf{c}) = \boldsymbol{\delta}$, and $\mathbf{\Phi}$ is the Jacobian matrix. Furthermore, the submatrix $\mathbf{\Phi}_{11}$ converges to the FTM.

3.2 Fast-Floquet Analysis

The fast-Floquet theory says that at $\psi = \Delta T$ the first blade is in the same position that was occupied by the second blade at $\psi = 0$. Hence, the state transition matrix computed for the time interval from $\psi = \Delta T$ to $\psi = 2\Delta T$ is identical to the one generated in the time interval from $\psi = 0$ to $\psi = \Delta T$ with the exception that the blade indices need to be permuted. Therefore, it is possible to find the transition matrix between any two instants that differ by ΔT from the transition matrix from $\psi = 0$ to $\psi = \Delta T$ in conjunction with the permutation matrix $\mathbf{P}$ (details to follow). Thus, the FTM can be generated by integrating the equations of motion through just ΔT. The general relation between any two instants $\psi = n\Delta T$ to $\psi = (n+1)\Delta T$ can be written as follows (Peters, 1995):

$$\mathbf{P}^n \mathbf{x}\left[(n+1)\Delta T\right] = \phi(\Delta T)\mathbf{P}^n \mathbf{x}\left[n\Delta T\right] \quad n = 0, 1, \ldots, Q-1 \tag{12}$$

From Eq. (12), we obtain

$$\mathbf{P}\mathbf{x}(\Delta T) = \mathbf{P}\phi(\Delta T)\mathbf{x}(0) \tag{13a}$$

$$\mathbf{P}^2\mathbf{x}(2\Delta T) = \left[\mathbf{P}\phi(\Delta T)\right]^2 \mathbf{x}(0) \tag{13b}$$

$$\mathbf{P}^Q\mathbf{x}(Q\Delta T) = \mathbf{x}(2\pi) = \left[\mathbf{P}\phi(\Delta T)\right]^Q \mathbf{x}(0) \tag{13c}$$

which leads to

$$\phi(2\pi) = \left[\mathbf{P}\phi(\Delta T)\right]^Q \tag{14}$$

Equation (14) is a crucially important relation in that it relates the FTM, $\phi(2\pi)$, with $\phi(\Delta T)$. Therefore, it is sufficient to compute $\phi(\Delta T)$ instead of $\phi(2\pi)$ because the Q-th power of $[\mathbf{P}\phi(\Delta T)]$ gives the FTM. Furthermore, the practical utility of Eq. (14) is keyed to the fact that the eigenvalues z_k of $\phi(2\pi)$ can be found from the eigenvalues $\overline{z}_k$ of $[\mathbf{P}\phi(\Delta T)]$ by the relation $z_k = \overline{z}_k^Q$. In fact, it is not even necessary to raise the eigenvalues $\overline{z}_k$ to the Q-th power because the modal damping levels and frequencies are computed by taking the logarithm of z_k; see Eq. (3). Therefore, we take the logarithm of $\overline{z}_k$ and simply multiply it by Q:

$$\overline{\sigma}_k = \frac{Q}{2\pi} \ln |\overline{z}_k| \tag{15a}$$

$$\overline{\xi}_k = \frac{Q}{2\pi} \arg(\overline{z}_k) = \frac{Q}{2\pi} \tan^{-1} \left(\frac{\mathrm{Im}(\overline{z}_k)}{\mathrm{Re}(\overline{z}_k)} \right) \tag{15b}$$

A comparison of Eqs. (15b) and (3b) is revealing in that the inherent frequency indeterminacy — addition of $\pm j\Omega$ in the conventional analysis versus $\pm j\Omega/Q$ in the fast-Floquet analysis — is reduced by a factor of Q. This is well borne out by the numerical results generated from these two analyses, which both use the mode-identification method of Nagabhushanam and Gaonkar (1995). Furthermore, Eqs. (14) and (15) show that the matrix $[\mathbf{P}\phi(\Delta T)]$ has one-to-one equivalence to the FTM; therefore, it is referred to as the equivalent FTM (EFTM).

In trimmed flight, the blades undergo periodic variations in the sectional angle of attack, which in turn produce periodic variations in the air velocity components and thus in rotor forces and moments. Therefore, for a rotor with identical blades, the contribution of each blade to the rotor forces and moments will be the same at a given azimuthal position because each azimuthal position has unique blade pitch and corresponding air velocity. Therefore, the number of blades in the rotor determines the period of oscillations of these forces and moments. Thus, it is required to rotate through ΔT so that a Q-bladed rotor has a blade in all azimuthal positions instantaneously. Hence, the variations of trim forces and moments in one period (T) can be described completely in ΔT of one rotor revolution. Therefore,

$$\left\{\begin{array}{c} C_T \\ C_d \\ C_l \\ C_m \end{array}\right\}_{\psi=0} = \left\{\begin{array}{c} C_T \\ C_d \\ C_l \\ C_m \end{array}\right\}_{\psi=\Delta T} \tag{16}$$

Similarly, the blade sectional circulatory lift influences the inflow forcing functions, which also can be described completely over the period ΔT. Therefore, the periodicity condition for inflow or wake states is

$$\left\{\begin{array}{c} \alpha_j^r \\ \beta_j^r \end{array}\right\}_{\psi=0} = \left\{\begin{array}{c} \alpha_j^r \\ \beta_j^r \end{array}\right\}_{\psi=\Delta T} \tag{17}$$

However, for the blade states, the rotor has to rotate through T for each blade to pass through all azimuthal positions. Thus, one complete rotor revolution is required to describe the blade states in trim. In other words, solutions over one complete revolution are required to establish the periodicity of the blade states. Nevertheless, it is possible to ascertain periodicity of blade states in ΔT of a revolution since all the blades follow the same trajectory as they go through one revolution with a phase shift of ΔT between the paths of each blade. Therefore, the states of an arbitrary q-th blade at an azimuthal position $\psi = \Delta T$ can be mapped onto the initial states of an identical $(q+1)$-th blade at $\psi = 0$. (The blade states can include structural states such as displacements and velocities, and blade-fixed aerodynamic states such as stall states.) Thus, for the periodicity of blade states, we have

$$\mathbf{x}_{b_{\psi=\Delta T}} = \mathbf{P}_b \mathbf{x}_{b_{\psi=0}} \tag{18}$$

where $\mathbf{x}_b$ is the $N_b \times 1$ vector of blade states defined as

$$\mathbf{x}_b = \lfloor \mathbf{x}_{\text{blade}\,1},\ \mathbf{x}_{\text{blade}\,2},\ \cdots, \mathbf{x}_{\text{blade}\,Q} \rfloor^T \tag{19}$$

In Eq. (18), $\mathbf{P}_b$ represents the $QN_b \times QN_b$ permutation matrix, which for an isolated rotor is given by

$$\mathbf{P}_b = \begin{bmatrix} \mathbf{0} & \mathbf{I}_b & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_b & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{I}_b \\ \mathbf{I}_b & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{20}$$

where $\mathbf{I}_b$ is the $N_b \times N_b$ unit matrix. Inclusion of body states requires minor changes in Eq. (20); for details, see Peters (1995).

Combining the blade states and the inflow states, we write

$$\{\mathbf{x}\}_{\psi_q = \Delta T} = \mathbf{P}\,\{\mathbf{x}\}_{\psi_q = 0} \tag{21}$$

In the above equation, the $N \times N$ permutation matrix $\mathbf{P}$ is given by

$$\mathbf{P} = \begin{bmatrix} \mathbf{0} & \mathbf{I}_b & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_b & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{I}_b & \mathbf{0} \\ \mathbf{I}_b & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{I}_w \end{bmatrix} \tag{22}$$

where $\mathbf{I}_w$ represents the unit matrix of size $N_w \times N_w$. As seen from Eqs. (16–21), the rotor trim forces, moments and periodic responses can be described in the interval ΔT. Therefore, Eq. (11), which improves the initial conditions that guarantee the periodic response as well as the unknown control inputs that satisfy the required flight conditions is modifed as (McVicar and Bradley, 1995; Peters, 1995)

$$\left\{ \begin{array}{c} \mathbf{x}(0) \\ \mathbf{c} \end{array} \right\}_{k+1} = \left\{ \begin{array}{c} \mathbf{x}_E(0) \\ \mathbf{c} \end{array} \right\}_{k} - \chi \begin{bmatrix} \mathbf{\Phi}_{11} - \mathbf{P} & \mathbf{\Phi}_{12} \\ \mathbf{\Phi}_{21} & \mathbf{\Phi}_{22} \end{bmatrix}^{-1} \left\{ \begin{array}{c} \mathbf{x}_E(\Delta T) - \mathbf{P}\mathbf{x}_E(0) \\ \delta \end{array} \right\} \tag{23}$$

Moreover, the matrix $\mathbf{P}^{\mathrm{T}}\mathbf{\Phi}_{11}$ converges to $[\mathbf{P}\ \phi(\Delta T)]$.

Thus far, we applied the concept of multiblade trim based on the fast-Floquet theory to a single-rotor model and showed that by exploiting the Q-fold symmetry it is feasible to obtain the trim and stability results from solutions over ΔT. This concept can be extended to ivestigate trim and stability of multirotor models where there are more than one rotor attached to a non-rotating structure (e.g., a coupled rotor-body system with one main rotor and one tail rotor). In such a system, each rotor rotates at a different angular velocity, and therefore, it is necessary to find the period of motion so that the solutions generated over this period can be used to predict the trim results and to compute the EFTM.

We consider a general system with R rotating substructures or interfaces. The period of motion for periodic solution can be established on the basis that the average angular velocities of all rotating interfaces are proportionate. In general, this holds good for rotors driven by an engine through gear trains with fixed gear ratios. Therefore, the relative angular velocity at each interface (Ω_i, $i = 1, 2, \cdots, R$) can be written as (Peters, 1995)

$$\Omega_0 \equiv \frac{\Omega_1}{M_1} = \frac{\Omega_2}{M_2} = \cdots = \frac{\Omega_R}{M_R} \tag{24}$$

where M_i ($i = 1, 2, \cdots, R$) represents the number of revolutions made by the i-th interface in the time interval $0 \leq t < T$, and T is the smallest common period, which is equal to $2\pi/\Omega_0$. For rotorcraft applications, the estimated common period T from Eq. (24) can be very large, rendering the numerical computations impractical. However, the period T can be reduced to an effective minimum by using the multiblade-trim concept. Since the rotors are attached to a non-rotating structure, each Ω_i in Eq. (24) is an absolute angular velocity of a rotor. If the i-th rotor has Q_i geometrically symmetric sectors, then $\overline{\Omega}_i = Q_i\Omega_i$ is the equivalent rotation rate for which $T_i = 2\pi/\overline{\Omega}_i$ or $\Omega_i T_i = 2\pi/Q_i$ gives a periodic sector position. Therefore, Eq. (24) can be rewritten as

$$\frac{\overline{\Omega}_1}{Q_1 M_1} = \frac{\overline{\Omega}_2}{Q_2 M_2} = \cdots = \frac{\overline{\Omega}_R}{Q_R M_R} \tag{25}$$

Now, multiplying Eq. (25) by the largest common factor of the denominators, say $Q_n M_n$, the largest common angular velocity of the multirotor system can be obtained as

$$\overline{\Omega}_0 = \frac{\overline{\Omega}_1}{N_1} = \frac{\overline{\Omega}_2}{N_2} = \cdots = \frac{\overline{\Omega}_R}{N_R} \tag{26}$$

In Eq. (26), N_i ($i = 1, 2, \cdots, R$) are the denominators obtained from $Q_n M_n / Q_i M_i$ ($i = 1, 2, \cdots, R$) and do not have any common factor. Hence, the smallest common period is $T = 2\pi/\overline{\Omega}_0$ or $\Omega_i T = \Delta T = 2\pi N_i / Q_i$ and N_i represents the number of sectors of the i-th rotor that pass through a given position during one common period. Thus, the use of the fast-Floquet theory can result in computational-time saving by a factor of M_iQ_i/N_i.

4 Parallel Fast-Floquet Analysis

4.1 Trim

Now, we present the parallel fast-Floquet algorithms for the trim analysis that are suitable for both the SIMD and MIMD architectures. To illustrate, we consider a single-rotor model with N structural and aerodynamic states, and c number of control inputs. The required extension to treat multirotor models is straightforward and is not spelled out explicitly. In the trim analysis, the bulk of the run time is for generating the $(N+c)\times(N+c)$ Jacobian in each iteration cycle. The conventional sequential algorithm generates these $(N+c)^2$ elements of the Jacobian one element at a time. By comparison, the parallel algorithms generate the elements of the Jacobian concurrently by dividing the computations suitably among the available processors. Specifically, the SIMD parallel algorithm generates the $(N+c)^2$ elements using $(N+c)^2$ processors; that is, each element of the Jacobian is generated by one processor. On the other hand, the MIMD algorithm generates each column of the Jacobian using one processor. To facilitate a better appreciation of these significant features, we begin with the algorithmic details of sequential shooting based on the fast-Floquet theory.

4.1.1 Sequential Fast Shooting

The sequential shooting algorithm centers on Eq. (11) and has the following seven instructions:

1. Assume $N+c=M$ arbitrary starting or initial values for the state variables of the augmented vector $\mathbf{s}$; that is, $N\times 1$ initial values for $\mathbf{x}(0)$ and $c\times 1$ initial values for the control-input vector $\mathbf{c}$.

2. Form the $N\times N$ permutation matrix $\mathbf{P}$ according to Eq. (22).

3. Perturb the M initial values *one initial value at a time* by a small amount ϵ_i, $i = 1, 2, \ldots, M$ and form $M+1$ vectors of starting values:

$$\mathbf{s} + \left\{ \begin{array}{c} \epsilon_1 \\ 0 \\ \vdots \\ 0 \end{array} \right\}, \mathbf{s} + \left\{ \begin{array}{c} 0 \\ \epsilon_2 \\ \vdots \\ 0 \end{array} \right\}, \cdots, \mathbf{s} + \left\{ \begin{array}{c} 0 \\ 0 \\ \vdots \\ \epsilon_M \end{array} \right\}, \mathbf{s} \tag{27}$$

4. Integrate Eq. (7) for $M+1$ times using the $M+1$ vectors of starting values through a time interval $\Delta T = 2\pi/Q$ and generate the solution vectors:

$$\mathbf{y}^i = \left\{ \begin{array}{c} \mathbf{x}(\Delta T) \\ \boldsymbol{\delta} \end{array} \right\}_{s+\epsilon_i} \quad \text{where } i = 1, 2, \ldots, M \text{ and } \mathbf{y} = \left\{ \begin{array}{c} \mathbf{x}(\Delta T) \\ \boldsymbol{\delta} \end{array} \right\}_s \tag{28}$$

where the vector $\boldsymbol{\delta}$ represents the trim error in satisfying Eq. (9). Moreover, the subscripts s and $s + \epsilon_i$, respectively, indicate the differences in the starting values; that is, one solution vector, $\mathbf{y}$, with starting-value vector $\mathbf{s}$ and M solution vectors, $\mathbf{y}^i$ $(i = 1, 2, \cdots M)$ with M vectors of perturbed starting values.

5. Form M columns of the Jacobian matrix Φ using

$$\left\{ \frac{\mathbf{y}^i - \mathbf{y}}{\epsilon_i} \right\}, i = 1, 2, \ldots, M \text{ or equivalently } \boldsymbol{\Phi} = \left[\begin{array}{cc} \boldsymbol{\Phi}_{11} - \mathbf{P} & \boldsymbol{\Phi}_{12} \\ \boldsymbol{\Phi}_{21} & \boldsymbol{\Phi}_{22} \end{array} \right] \tag{29}$$

where $\mathbf{P}^{\mathrm{T}}\boldsymbol{\Phi}_{11}$ converges to $\mathbf{P}\phi(\Delta T)$.

6. Generate the error vector $\mathbf{E}^k$. Specifically, at the k-th iteration counter

$$\mathbf{E}^k = \left\{ \begin{array}{c} \mathbf{x}(\Delta T) - \mathbf{P}\mathbf{x}(0) \\ \boldsymbol{\delta} \end{array} \right\}^k \tag{30}$$

where $\mathbf{x}(\Delta T)$ represents the solution vector at the end of period ΔT and $\boldsymbol{\delta}$ is the trim-error vector corresponding to the initial-condition vector $\mathbf{s}$.

7. Improve the solution with Newton damping parameter χ:

$$\mathbf{s}^{k+1} = \mathbf{s}^k - \chi \boldsymbol{\Phi}^{-1} \mathbf{E}^k \tag{31}$$

Instructions 3–7 are repeated till the convergence of control inputs and periodic responses.

4.1.2 SIMD-Parallel Fast Shooting

As seen from Eqs. (27–31), each instruction is applied to a large number of input data and the operations on these data are independent. The SIMD-parallel fast shooting algorithm exploits this independence in such a way that all the processors execute the same instruction but on different data. Given this background, we present the SIMD algorithm with seven instructions in a format that is implemented.

1. Assume an arbitrary vector $\mathbf{s} = \lfloor s_1\, s_2 \cdots s_M \rfloor^T$ of starting values for trim results of periodic responses and control inputs. Then replicate the vector $\mathbf{s}$ $(M+1)$ times to form a matrix of size $M \times (M+1)$:

$$\begin{bmatrix} s_1 & s_1 & \cdots & s_1 & s_1 \\ s_2 & s_2 & \cdots & s_2 & s_2 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ s_M & s_M & \cdots & s_M & s_M \end{bmatrix} \tag{32}$$

2. According to Eq. (22), form the $N \times N$ permutation matrix $\mathbf{P}$ in such a way that each element of $\mathbf{P}$ is stored in one processor.

3. Perturb the matrix in Eq. (32) (excluding the last column of the matrix) along the leading diagonal by a small amount ϵ_i, $i = 1, 2, \ldots, M$, and generate the $M \times (M+1)$ matrix of initial conditions:

$$\mathbf{X} = \begin{bmatrix} s_1+\epsilon_1 & s_1 & \cdots & s_1 & s_1 \\ s_2 & s_2+\epsilon_2 & \cdots & s_2 & s_2 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ s_M & s_M & \cdots & s_M+\epsilon_M & s_M \end{bmatrix} \tag{33}$$

4. Using the above $M \times (M+1)$ matrix of initial conditions, integrate Eq. (7) in parallel and generate the solution matrix $\mathbf{Y}$ at the end of period $\psi = \Delta T = 2\pi/Q$:

$$\mathbf{Y} = \begin{bmatrix} y_1^1 & y_1^2 & \cdots & y_1^M & y_1^{M+1} \\ y_2^1 & y_2^2 & \cdots & y_2^M & y_2^{M+1} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ y_M^1 & y_M^2 & \cdots & y_M^M & y_M^{M+1} \end{bmatrix} = \left[\mathbf{y}^1\, \mathbf{y}^2 \cdots \mathbf{y}^M\, \mathbf{y}^{M+1}\right] \tag{34}$$

The integration of equations of motion starts at $\psi = 0$ and proceeds through an interval $\psi = \Delta T$ with a finite number of azimuthal positions as integration steps; that is, $\psi_0 = 0 \rightarrow \psi_1 = \psi_0 + \Delta\psi \rightarrow \cdots \rightarrow \psi_n = \Delta T$. The operational details at each azimuthal position are keyed to the parallel computations associated with the equations of motion and trim; for details, see Subramanian (1996). Thus, the computations of the error in satisfying Eq. (9) and the $(M+1)$ sets of integrations for the $M \times (M+1)$ matrix of initial conditions are carried out in parallel, and thereby each element of the solution matrix $\mathbf{Y}$ is computed by one processor. In other words, the $M(M+1)$ elements of the matrix $\mathbf{Y}$ are generated by using $M(M+1)$ processors.

5. Form the Jacobian matrix $\mathbf{\Phi}$ according to Eq. (29).

6. Estimate the error using Eq. (30).

7. Improve the solution following Eq. (31).

The instructions 3–7 are repeated till convergence of trim results.

4.1.3 MIMD-Parallel Fast Shooting

The MIMD-parallel fast shooting algorithm follows a self-scheduling algorithm utilizing the master-slave processor approach. In general, the master processor executes the main program, which contains the entire algorithm and the calls to communication routines for assigning the tasks to the slave processors. The slave processors run a segment of the main program, which contains the routines that are necessary for accomplishing the tasks assigned by the master processor. Thus, the algorithm is executed sequentially on the master processor until a parallelized step is reached. At that step, the master processor breaks up the problem into a number of independent subprograms and assigns them to the slave processors. Each slave processor completes the task assigned to it and returns the results to the master processor. The master processor then assembles these results and proceeds to the next step in the algorithm. Specifically, in the trim analysis, the master processor sets up the initial conditions for integration, distributes them to the slave processors, forms the Jacobian matrix using the solutions received from the slave processors, and upgrades the trim values of control inputs and initial conditions for periodic responses at the end of each iteration. Similarly, each slave processor receives a set of initial conditions from the master processor, integrates the equations of motion, estimates the forces and moments, and returns this information to the master processor. Thus, in each iteration cycle, each column of the Jacobian is generated by one slave processor for $(M+1) < p_s$ (p_s = number of slave processors). For $(M+1) > p_s$, this process is repeated till all the columns of Jacobian are generated. These operations are carried out in two parts; the first part corresponds to the master processor and the second to the slave processors, and the instructions in the respective parts are as follows:

Master Part:

1. Assumes a vector s of size $N + c = M$, which represents a set of arbitrary initial conditions for structural and aerodynamic states as well as for control inputs.

2. Forms the permutation matrix $\mathbf{P}$.

3. Forms $(M+1)$ sets of initial-condition vectors by perturbing only one element of $\mathbf{s}$ for each set of initial conditions. The $(M+1)$-th set is a vector of unperturbed states; see Eq. (27).

4. Sends as many sets of initial conditions as the available number of slave processors. For example, if $(M+1) > p_s$, it sends the first p_s sets of initial condition vectors.

5. Receives the solution vectors $\mathbf{y}^i$ ($i = 1, 2, \ldots M$) or $\mathbf{y}$ (see Eq. (28)) at the end of the period ΔT from the slave processors and stores them.

6. Checks whether all sets of initial-condition vectors are processed. If not, identifies the slave processors that have completed their tasks and distributes the remaining sets of initial conditions till all the columns of the Jacobian are generated. Otherwise, sends an end signal to the slave processors.

7. Forms the Jacobian matrix $\mathbf{\Phi}$.

8. Generates the error vector $\mathbf{E}_k$.

9. Improves the solution according to the Newton iteration.

10. Repeats steps 3–9 till convergence.

Slave Part:

1. Receives one set of initial conditions for the structural and aerodynamic states and control inputs from the master processor.

2. Integrates Eq. (7) through the time interval ΔT and generates the solutions vector $\mathbf{y}^i$ ($i = 1, \cdots, M$) or $\mathbf{y}$ (see Eq. (28)). Thus, each column of the Jacobian is generated in parallel by one processor although the M elements of each column are generated sequentially.

3. Sends the solution vector $\mathbf{y}^i$ or $\mathbf{y}$ and the node details.

4. Repeats steps 1–3 till an end signal is received from the master processor and then exit to the master part.

4.2 Stability

The stability results comprise the modal damping levels and frequencies from the eigenvalues of the EFTM; see Eqs. (15a) and (15b). It can be seen from Eq. (15b) that the frequency is determined from the inverse arctangent function, which results in multiple values. Therefore, to compute the frequencies and thereby to identify the modes, we follow the method of Nagabhushanam and Gaonkar (1995) with the modification of replacing the eigenvalues and eigenvectors of the FTM by those of the EFTM; see Eq. (23). For each eigenvector, say $\mathbf{x}_i$, we compute the complex ratio of the derivative $\dot{x}_i$ and the state x_i corresponding to the most dominant component. The imaginary part of this ratio with a suitable correction, which is an integer multiple of Q, closely approximates the frequency of the mode. In this study, an LAPACK eigenvalue subroutine DGEEV for real unsymmetric matrices is used in the SIMD-parallel algorithm to compute the eigenvalues and eigenvectors of the EFTM (Anderson et al., 1992). Similarly, a ScaLAPACK eigenvalue subroutine developed for MIMD-type computing offers considerable promise for the EFTM eigenanalysis (Blackford et al., 1997).

5 Computational Reliability

The trim and stability predictions are computationally demanding and require solutions for nonlinear differential equations of motion coupled with algebraic transcendental equations of trim. These solutions involve a large number of numerical operations such as integrations of equations, linearizations for Newton improvement and iterations with improved starting values. Moreover, these computations are susceptible to numerical corruptions, which can get magnified in the final results of trim and stability predictions. Therefore, it becomes imperative to have a means of quantifying the computational reliability.

As seen from Eqs. (29) and (31), the Jacobian influences the convergence of trim results in each iteration cycle. Moreover, the EFTM is extracted from the Jacobian in the converged cycle, and the stability results of damping levels and frequencies are computed from the eigenanalysis of the EFTM. Therefore, computational reliability concerns both trim and stability analyses. To this end, we introduce three computational reliability parameters:

1. The condition number of the Jacobian matrix $\mathbf{\Phi}$:

   By definition

$$\text{Cond.}(\mathbf{\Phi}) = \frac{\left[\text{max. eigenvalue of}\,\mathbf{\Phi}^T\mathbf{\Phi}\right]^{\frac{1}{2}}}{\left[\text{min. eigenvalue of}\,\mathbf{\Phi}^T\mathbf{\Phi}\right]^{\frac{1}{2}}} \tag{35}$$

2. The condition number Cond.(λ) for the eigenvalue of the mode of interest:

   Let $\mathbf{x}$ and $\mathbf{y}$ represent the right and left eigenvectors of the EFTM corresponding to an eigenvalue λ; that is,

$$[\text{EFTM}]\mathbf{x} = \lambda\mathbf{x} \text{ and } [\text{EFTM}]^{\text{T}}\mathbf{y} = \lambda\mathbf{y} \tag{36}$$

   Then, the condition number of λ is computed from the expression

$$\text{Cond.}(\lambda) = |\mathbf{y}^T\mathbf{x}|^{-1} \tag{37}$$

3. The residual error for the eigenpair $(\lambda, \mathbf{x})$ of the mode of interest:

   It is given by

$$\varepsilon = \frac{\|[\text{EFTM}]\mathbf{x} - \lambda\mathbf{x}\|}{\|\lambda\mathbf{x}\|} \tag{38}$$

For additional details, see Ravichandran et al., (1990).

6 Parallel Performance Metrics

Unlike serial computing, measuring the effectiveness of parallel implementations is an important aspect of parallel computing. Ideally, a perfect parallel algorithm executing on p processors is expected to reduce the uniprocessor run time by a factor p. Following the literature (e.g., Karp and Flatt, 1990), we introduce five performance metrics, which collectively provide a measure of the performance of the parallel fast-Floquet analysis. The first metric is the growth of run time with the order and number of processors; this is a directly measured metric and the easiest to interpret. By definition, the parallel run time is the total elapsed time from the beginning of the parallel program to the end, till all the computations are completed. The other three metrics are speedup, sequential fraction and efficiency, which are derived from the measured run time, number of processors used and predicted uniprocessor run time. We emphasize that, in general, the individual processors of SIMD systems are less powerful and therefore, a direct measurement of uniprocessor run time becomes impractical for large models. Moreover, the master-slave processor approach for the MIMD-parallel algorithm requires at least two processors. Finally, we address the issue of portability of parallel fast-Floquet analysis.

Speedup S_p provides a measure of how a parallel algorithm executing on p processors compares with itself executing on one processor. With t_j representing the parallel run time on j processors, speedup is defined as

$$S_p = \frac{t_1}{t_p} \leq p \tag{39}$$

Similarly, efficiency E_p measures how effectively the processors are used; in other words, how busy they are kept relative to each other:

$$E_p = \frac{S_p}{p} \leq 1 \tag{40}$$

An efficiency approaching one means we are "getting to the best" that the processors can do (Kumar et al., 1994). Equations (39) and (40) show that the results of S_p and E_p need to be interpreted in a relative sense since they depend on the run time constraint as the model order increases and on whether the number of processors is fixed or varied. In other words, the results of speedup and efficiency provide a means of compromising between how fast the job needs to be completed and how the processors are kept busy.

Following Morse (1994), we predict the uniprocessor run time t_1 in Eq. (39). Basically, t_1 is expressed as a sum of sequential portion t_{1_s} and parallel portion t_{1_p}:

$$t_1 = t_{1_s} + t_{1_p} \tag{41}$$

The assumptions involved in Eq. (41) are that the problem can be divided completely into a sequential part and a parallel part, and that the sequential part and overhead such as due to interprocessor communication are independent of the problem size. Moreover, the parallel part is perfectly parallel and can be divided equally among the processors. Since only the parallel part can be speeded up, the run time with p processors is

$$t_p = t_{1_s} + \frac{t_{1_p}}{p} \tag{42}$$

The above equation is used to predict the uniprocessor run time $t_1 (= t_{1_s} + t_{1_p})$. Basically, we run the same job for different values of p and measure the corresponding parallel run times t_p. Using t_p, the sequential portion t_{1_s} and parallel portion t_{1_p} are computed by a least-square approach. Now, we express t_{1_s} and t_{1_p} in terms of dimensionless serial fraction f, which by definition is:

$$t_{1_s} = f t_1 \text{ and } t_{1_p} = (1-f)t_1 \tag{43}$$

Therefore, in Eq. (42), the parallel run time t_p can be expressed as

$$t_p = f t_1 + \frac{(1-f)t_1}{p} \tag{44}$$

and thereby the speedup in Eq. (39) can be rewritten in the form

$$S_p = \frac{1}{f + \frac{1-f}{p}} \leq \frac{1}{f} \tag{45}$$

Equation (45) is the celebrated Amdahl's law and shows the importance of serial fraction on the overall effectiveness of parallel computing (Kumar et al., 1994). Moreover, from Eq. (45), the serial fraction f can be expressed in terms of speedup S_p and number of processors p:

$$f = 1 - \frac{1 - 1/S_p}{1 - 1/p} \tag{46}$$

In general, the serial fraction is strictly not independent of the number of processors, and moreover, p takes on only integer values. Nevertheless, substituting Eq. (45) in (40) and differentiating $1/E_p$ with respect to p, we get

$$\frac{d}{dp}\left(\frac{1}{E_p}\right) = f \tag{47}$$

Portability is a performance measure that cannot be quantified; nevertheless, it is very important in practical parallel computing. Portability is the ease with which the same parallel algorithm can be implemented on different machines/architectures. Due to architectural differences, the same parallel fast-Floquet algorithm can not be implemented on both SIMD and MIMD parallel computers. Moreover, a particular implementation on a SIMD parallel computer (e.g., MasPar MP-1) would require substantial modifications for other SIMD computers, which will lead to machine dependent codes. However, the MIMD parallel fast-Floquet algorithm can be ported among different MIMD parallel computers since the algorithm employs the MPI (Snir et al., 1996). It is a standard message passing library for interprocessor communication, which facilitates development of portable algorithms as demonstrated by the implementation of the same parallel fast-Floquet analysis on both the massively parallel computer and distributed computing system of networked workstations.

7 Results

We primarily address the performance metrics of the parallel fast-Floquet analysis with a passing reference to the computational reliability parameters. Specifically, the performance metrics comprise the run-time variation with the model order and number of processors as well as speedup, efficiency and serial and parallel fractions. The run time refers to the total elapsed time for the trim and stability analyses; nearly 99% of it is for the trim analysis. Similarly, the computational reliability parameters comprise the condition number of the Jacobian in the converged cycle, condition number of the eigenvalue for the lag regressing mode and residual error of the corresponding eigenpair. All serial computations are done on a VAX 4320. The parallel computations are done on three types of hardware: a MasPar MP-1 with 8192 processors (SIMD), an IBM SP-2 with 140 processors (MIMD) and a network of 13 SUN SPARC stations (MIMD-based distributed computing). Since the distributed computing system is also based on the MIMD architecture, the same algorithm is used for the IBM SP-2 and networked workstations.

To generate the results we use models of hingeless rotors in trimmed flight. The rotors have three and more blades executing rigid flap and lag motions and are isolated in that their support systems are stationary. The ONERA dynamic stall models of lift and drag are used to represent the airfoil aerodynamics, and the rotor downwash dynamics is represented by a finite-state three-dimensional wake model. The model order or number of states is controlled by varying the number of wake harmonics or wake states in modeling the wake, the number of aerodynamic elements per blade in representing the stall dynamics and the number of blades. Unless otherwise stated, the following baseline parameters are used: $\mu = 0.3$, $\gamma = 5$, $P_\beta = 1.15$, $\omega_\zeta = 1.14$, $\sigma = 0.05$, $a = 6.28$, $C_{d_0} = 0.0079$, $C_w = 0.00375$, and $\overline{f} = 0.01$. The results are presented in three phases according to the type of hardware used; Figs. 1–4 and 5–8, respectively, refer to the massively parallel SIMD and MIMD computers, and Figs. 9–12 to a distributed computing system of networked workstations. Finally, Fig. 13 gives a summary of the run-time variation with the order for the fast-Floquet analysis on the serial and three types of parallel hardware. To help appreciate the results we recall that the standardization of the parallel performance metrics is still evolving and distributed computing in particular represents an emerging area of the past few years. Moreover, evaluation and interpretation of these metrics depend upon several variables such as type of problem and computing hardware. As seen from Eqs. (39) and (40), the speedup S_p and efficiency E_p are not measured metrics but predicted metrics that depend not only on the measured values of the run time t_p but also on the predicted uniprocessor run time t_1 for the parallel algorithm. Prediction of t_1 in turn is based on the assumption typified by Eq. (42) and on a series of measured values of t_p; this requires running the same job for different values of p and then predicting t_1 by a least-square approach. This means t_1, S_p and E_p are very much dependent on the problem and hardware, and S_p and E_p in particular are sensitive to the accuracy of predicting t_1. Therefore, even when the same algorithm is used on two different hardwares, considerable care needs to be exercised in comparing the two sets of metrics. Despite such drawbacks, these metrics along with the measured values of run time and its variation with the order and number of processors provide a broad assessment of how fast the job is completed (quantified by S_p) and how well it is achieved (quantified by E_p) on that specific hardware by that specific parallel code.

Figure 1 shows a comparison of the parallel run times for the conventional- and fast-Floquet analyses. The model order varies from 94 to 430. The fast-Floquet analysis provides nearly a Q-fold reduction in run time throughout. As an example, for a model with $M = 376$ and $Q = 3$, the observed fast-Floquet run time is 5667 seconds; this compares reasonably with the expected Q-fold reduction: conventional-Floquet run time of $12812/5667 \approx 2.3$. For clarity of graphical presentation the expected reduction is shown for only four discrete cases of a three-bladed rotor with $M = 227$, 262, 301 and 376. Moreover, both analyses show that parallel run time varies with jump-type increases occurring around $M = 128, 192, \cdots$, etc. Specifically, the run time remains nearly constant within the intervals $64 \leq M \leq 128$, and $129 \leq M \leq 192$ for relatively small order models, and linearly varies with a small slope within the intervals $193 \leq M \leq 256$, and $257 \leq M \leq 384$ for relatively large order models. Such a variation is due to mapping of variable arrays onto the processor array of MasPar and the consequent increase in communication overhead in accessing data stored in different memory layers; for details on memory management, see the discussion by Subramanian and Gaonkar (1996).

The widely used parallel-performance measure is speedup, S_p; it is shown in Fig. 2. With increasing model order, it generally increases although there is some unexpected decrease from $M = 376$ to $M = 430$. Overall, Fig. 2 shows the effectiveness of the parallel fast-Floquet analysis in reducing the uniprocessor run time. To illustrate, we have a speedup of 2000 for $M = 376$; this means the predicted uniprocessor run time is 2000 times the observed parallel run time. However, as seen from Eqs. (39) and (40), the speedup and efficiency depend upon the number of processors and are to be interpreted in a relative sense; that is, how well the processor underutilization or idle time is minimized while the computations are sped up. Therefore, for a complete picture we need to study simultaneously the variations of the speedup, efficiency and run time as we increase the model size or order with the number of processors. This is taken up in the next figure.

Figure 3 shows these variations for four models of progressively increasing order: $M =$ 94, 262, 376 and 430. In MasPar MP-1, we can run a job by using a specified number of processors p ($p = 1024$, 2048, 4096 and 8192); if not stipulated by the user, p is fixed through a compiler directive. As seen from Fig. 3, for a model of fixed order, we can reduce the run time by increasing the number of processors and thereby increasing the speedup. However, this is accompanied by a reduction in efficiency. To illustrate, we consider a model with $M = 376$ and use 2048 processors, for which the run time is 9872 seconds, and the speedup and efficiency are, respectively, 1147 and 56%. By doubling the number of processors to 4096, we reduce the run time to 7354 seconds, a reduction of about 26%, and increase the speedup to 1540, an increase of about 34%. But the efficiency comes down to 38%. Thus in summary, Fig. 3 shows how the run time can be controlled by varying the number of processors with the order; that is, by a judicious combination of speedup and efficiency. It also shows how parallelism provides a routine means of removing the barrier of run-time constraint of the Floquet analysis with hundreds of states. The less-than-excellent ($< 60\%$) efficiency figures seem to indicate that the load balancing of the SIMD algorithm merits further improvement and that the simple rigid flap-lag model is not 'big enough' to properly exploit the available computing power even with $p = 1024$.

In Fig. 4, the serial fraction f and the parallel fraction $1-f$ are presented with increasing model order M. The serial fraction is an indirect measure of efficiency; see Eq. (47). It essentially decreases (or equivalently the parallel fraction increases) with increasing model order. As an example, the serial fraction is about 0.00081 for $M = 94$ and it decreases to 0.00039 for $M = 376$. This means, according to Amdahl's law (Eq. 45), the upper bound on speedup S_p increases from 1240 to 2568. This is expected because in the Floquet analysis the bulk of the run time is for repeated integrations, which increase with the order. Since the parallel analysis performs these integrations independent of each other, its effectiveness increases with increasing order.

Figures 5–8 show the results based on the MIMD algorithm implemented on a MIMD computer for three models of order $M = 227$, 329 and 395. In particular, Fig. 5 shows the variation of the run time with the number of processors for $2 \leq p \leq 64$. Overall, as expected, the run time decreases with increasing number of processors for a fixed model order and increases with increasing model order for a fixed number of processors. In particular, for the model with $M = 227$, the run time remains nearly constant for $p > 10$; that is, a further increase in the number of processors yields no appreciable saving in run time. For the larger models with $M = 329$ and 395, the rate of reduction in run time with p is significant only for $p \leq 16$, and the run time virtually flattens out for $p > 32$ or so. This lack of reduction in run time with increasing p is associated with delays in interprocessor communication. In IBM SP-2, communication takes place through a switch, which is much slower when compared to the speed of individual processors. Thus, as the number of processors increases, the communication delays increase accordingly. This negates any computational gains achieved through parallelism. The impact of this communication overhead on speedup and efficiency is studied in Fig. 6, where the results are presented for $2 \leq p \leq 32$.

In Fig. 6, while part 'a' shows the variation of speedup with the number of processors and model order, part 'b' shows the corresponding variation of efficiency. As expected, for a fixed order with increasing number of processors, the speedup increases and efficiency decreases. However, for a fixed number of processors, both speedup and efficiency increase with increasing order M. This means if the job needs to be completed faster, it is necessary to increase the number of processors; see also Fig. 5. But this increase in speedup is accompanied by reduced efficiency. Figure 6 also shows that the speedup and efficiency figures are close to the ideal for $p \leq 6$ for $M = 329$ and 395. For example, the speedup and efficiency are, respectively, equal to 1.8 and 95% for $M = 395$ with $p = 2$, and with $p = 10$, a five-fold increase, the speedup increases to 7.5 and the efficiency comes down to 75%. However, in general, Fig. 6 shows that S_p and E_p deviate considerably from the ideal values with increasing number of processors ($p > 10$). This is related to idling of the processors and interprocessor communication. In the master-slave algorithm, the slave processors, which are larger in number, remain idle when the master executes the serial portion. Similarly, the master remains idle when the slaves are performing the parallel portion of the problem. Consequently, the estimated uniprocessor run time is dominated by the serial portion of the problem, which limits speedup; see Eqs. (43) and (45). Moreover, the computations involved in the simple rigid blade model is not large enough to fully exploit the enormous computing power of the SP-2 computer with $p > 10$.

In Figs. 5 and 6, the nearly flat variation in run time and speedup for $p > 32$ or so merits some additional comments. In the present MIMD version, a wide range of operations is carried out serially in the master processor and these operations are explicitly identified in the master-part of the MIMD-parallel fast shooting algorithm in Section 4.1.3. The key point here is that during such serial executions the slave processors are kept idle. Although the bulk of these operations is serial, a portion of these may lend to further parallelization, which, in turn, reduces the number of serial operations and provides a better utilization of the slave processors (Strawn 1997). This further parallelization offers promise and investigation is continuing.

The next two figures address the overall effectiveness of the parallel fast-Floquet analysis based on the serial and parallel fractions (Fig. 7) and on the rate of change of reciprocal of efficiency with respect to the number of processors (Fig. 8); see also Eqs. (43), (45) and (47). Recall that the serial fraction limits the upper bound of speedup and can be an indirect measure of efficiency. As seen from Fig. 7, with increasing model order the serial fraction decreases or equivalently the parallel fraction increases. For example, the serial fraction, which is about 0.18 for $M = 227$ decreases to 0.045 for $M = 395$. Therefore, the upper bound of speedup increases from 5.5 to 20. This means the degree of parallelism increases with increasing M. This is expected as well; in the parallel fast-Floquet analysis, the bulk of the saving in run time is achieved by performing the repeated integrations in parallel, and the number of these integrations increases with increasing order. Similarly, Fig. 8 shows that the $1/E_p$-versus-p curve is approximately linear for all three models and that the slope of the curve decreases with increasing order. In other words, the serial fraction decreases and consequently efficiency increases with increasing M; see also Fig. 6.

In Figs. 9–12, we present the results from a distributed computing system. The computations are carried out on a network of 13 low-end SUN SPARC stations (IPC and LX). These workstations are heterogeneous in that the individual processors differ in memory and clock speed. Such architectural differences are dealt with efficiently by dynamically balancing the load among the processors through the master-slave processor algorithm. The workstations are accessible through a department-wide network; it is a local area network (LAN), which uses ethernet communication channel.

Figure 9 shows how the run time varies as we increase the number of processors and model order. For illustration, we consider the same three models treated earlier with $M = 227$, 329 and 395. As expected, the run time for a given model decreases with increasing number of processors and increases with increasing order. These features are similar to those observed on IBM SP-2 in Fig. 5, except that the run times are much longer. Moreover, the reduction in run time is appreciable, say for $p < 7$, for the model with $M = 395$ and to a lesser extent for the other two models. For $p > 7$, the run time for all three models either remains nearly constant or decreases only slightly. This means increasing the number of processors beyond a certain value does not yield a significant reduction in run time owing to communication overhead, which increases with increasing number of processors. In the distributed computing system of networked workstations, the interprocessor communication is through the ethernet communication channel, which has a small bandwidth (typically a few Mbits/sec) that remains the same even when more workstations are added to the network. By comparison, the processors of these workstations are sufficiently powerful in performing a few MFLOPS. Therefore, for large models with increasing number of processors, the communication channel can become a bottleneck with limited communication bandwidth and high latency. Moreover, in the master-slave algorithm, the master processor communicates with the slave processors $(M + 1)$ times, and the length of the message to be communicated also increases with M. Furthermore, in a network of workstations, it is not always possible to perform the computations in a 100% dedicated fashion since the workstations have slight loads due to non-computational related operations, and the network is still being used by other workstations that are not actually participating in the computations. Thus, the overall performance of the distributed computing system is lower than what the system can nominally deliver. Nevertheless, we emphasize that the parallel code is completely portable from the massively parallel SP-2 computer to the distributed computing system and vice versa without any modification whatsoever.

The effectiveness of the parallel fast-Floquet analysis on a distributed computing system is further addressed in the next three figures, which, respectively, show the variation of the speedup and efficiency with increasing number of processors (Fig. 10), the serial and parallel fractions with increasing model order (Fig. 11), and $1/E_p$ curve with increasing number of processors (Fig. 12). As seen from Fig. 10, for a fixed model order, while the speedup increases with increasing number of processors, the efficiency basically decreases. Moreover, both speedup and efficiency increase with increasing order for a fixed number of processors. These results are similar to those obtained from IBM SP-2 computer (*e.g.*, Fig. 6). In particular, both speedup and efficiency are close to the ideal values for $M = 395$ with $p \leq 7$, which is also the case for other two models for $p \leq 5$. Figure 11 basically shows that the parallel fraction is greater than 0.92 for the models considered; in fact, it is about 0.97 for $M = 395$ and consequently the serial fraction is a small number. This indicates that the fast-Floquet analysis is tailored to distributed computing as well. Figure 12 essentially corroborates the finding of Fig. 8 in that the slope of the $1/E_p$-versus-p curve decreases with increasing M, indicating that efficiency increases with M. However, it is seen that this curve has localized deviations from linearity, which merit further investigation.

Of particular importance is a comparison of how the run times grow with the order for the serial and parallel analyses, both based on the fast-Floquet theory. This comparison provides a better appreciation for the results thus far presented and for the necessity of turning to parallelism. The final figure does just that. As seen from Fig. 13, the serial run time grows between quadratically and cubically with the order ($\approx M^{2.4}$). Owing to this rapid growth the presentation is restricted to relatively small-order models ($94 \leq M \leq 169$). For example, the run time, which is 6 hours and 45 minutes for $M = 94$, increases to 2 days and 12 hrs for $M = 169$. This is despite the fact that the fast-Floquet theory already provides nearly a Q-fold reduction in run time. Thus, the serial fast-Floquet analysis is not attractive for the routine treatment of models with hundreds of states. We also see the dramatic impact of parallelism on the run time and its growth with the order, which varies from 79 to 376. It is true that the sequential and parallel run times do not permit a direct comparison. Nevertheless, the comparison shows that the wall-clock time for job completion is dramatically reduced. Furthermore, the rate of growth of the parallel run time with the order is milder compared to the formidable growth of the serial run time. (For the scale adopted, localized jumps of the type in Fig. 2 are masked.) It is also instructive to compare the run time from the three parallel implementations; this is done in the inset of Fig. 13, which shows the finer details of the run-time variation with model order. For example, for $M = 360$, the serial run time runs into weeks (not shown) whereas the parallel run times are 630, 5400 and 8100 seconds for the MIMD, SIMD and distributed computing systems, respectively. Two additional features merit mention. First, compared to the other two parallel implementations, the implementation on IBM SP-2 takes much less run time; this is primarily due to the fact that the individual processors are extremely powerful and, compared to the distributed computing system communication network, much faster. Second, the run times from the implementations on MasPar MP-1 and distributed computing systems are comparable (We mention in passing that MasPar is built with less powerful processors with limited memory and speed.) Nevertheless, the point is that the distributed computing system delivers performance fairly comparable to that of high-cost massively parallel SIMD computers and to some extent to that of the MIMD computer.

Table 1 presents a sample of computational reliability parameters for the parallel fast-Floquet analysis from the IBM SP-2 computer and distributed computing system. It is seen that the reliability parameters from both the implementations are comparable. Moreover, the condition numbers of the Jacobian as well as the eigenvalue condition numbers are acceptable and the residual errors of the eigenpairs are negligible; for additional details, see Achar and Gaonkar (1993), and Ravichandran et al. (1990).

8 Conclusions

Thus far, the serial and parallel algorithms of the fast-Floquet analysis are presented for the trim and stability predictions; the fast-Floquet analysis exploits the Q planes of symmetry that exist for a rotor with Q identical and equally spaced blades. In particular, while the same MIMD algorithm is implemented on the MIMD and distributed computing systems, the SIMD algorithm is implemented on the SIMD system. Large models with hundreds of states are treated, and a comprehensive database on parallel performance and computational reliability is generated. A major finding follows: Concerning the turnaround time, and algorithm development and implementation, the treatment of large models on a distributed computing system is as routine as that of relatively small models on a serial computer. Other specific findings are:

1. These serial and parallel algorithms reduce the run time and frequency indeterminacy of the corresponding conventional Floquet analysis by nearly a factor of Q. Despite

the reduction, the run time of the serial algorithm grows between quadratically and cubically with the order; this still limits its utility to relatively small-order models. By comparison, in both the parallel algorithms, the run time and its growth with the order can be controlled by a judicious combination of speedup and efficiency, which, respectively, show how fast the job is completed with p processors and how equally the loads on these p processors are distributed.

2. The serial and parallel fractions of the SIMD and MIMD algorithms are fairly comparable. Moreover, the speedup and efficiency figures are close to the ideal values, that is, the speedup close to p and nearly 90% efficiency for the MIMD algorithm for some combinations of model order M and number of processors p. Throughout, these figures are much lower than the ideal values for the SIMD algorithm. Thus, these parallel-performance data demonstrate the suitability of the parallel fast-Floquet analysis on a distributed computing system.

9 References

Anderson, E., et al., *LAPACK Users's Guide*, Society for Industrial and Applied Mathematics Publication, Philadelphia, 1992.

Achar, N. S. and Gaonkar, G. H., 1993, "Helicopter Trim Analysis by Shooting and Finite Element Methods with Optimally Damped Newton Iterations," *AIAA Journal*, Vol. 31, pp. 225-234.

Achar, N. S. and Gaonkar, G. H., 1994, "An Exploratory Study of a Subspace Iteration Method as an Alternative to the QR Method for Floquet Eigenanalysis," *Journal of Mathematical and Computer Modeling*, Vol. 19, pp. 69-73.

Blackford, S. L., et al., 1997, *ScaLAPACK Users's Guide*, Society for Industrial and Applied Mathematics Publication, Philadelphia, 1997.

Chunduru, S. J., 1995, "*Dynamic Stall and Three-Dimensional Wake Effects on Trim, Stability and Load of Hingeless Rotors with Fast Floquet Theory,* Ph.D. Thesis, College of Engineering, Florida Atlantic University, Boca Raton, FL.

Chunduru, S. J., Subramanian, S. and Gaonkar, G. H., 1997, "Dynamic Stall and Three-

Dimensional Wake Effects on Trim and Stability of Hingeless Rotors with Experimental Correlation," *Journal of the American Helicopter Society*, Vol. 42, pp. 370-382.

Gaonkar, G. H. and Peters, D. A., 1987, "Review of Floquet Theory in Stability and Response Analysis of Dynamic Systems with Periodic Coefficients," The R. L. Bisplinghoff Memorial Symposium Volume on Recent Trends in Aeroelasticity, Structures and Structural Dynamics, University Presses of Florida, pp. 101–119.

Karp, A. H. and Flatt, H. P., 1990, "Measuring Parallel Processor Performance," *Communications of the Association for Computing Machinery*, Vol. 33, pp. 539-543.

Kumar, V., Grama, A., Gupta, A. and Karypis, G., 1994, *Introduction to Parallel Computing, Design and Analysis of Algorithms*, Benjamin/Cummings Publishing Company, New York.

McVicar, J. S. G. and Bradley, R., 1995, "Robust and Efficient Trimming Algorithm for Application to Advanced Mathematical Models of Rotorcraft," *Journal of Aircraft*, Vol. 32, pp. 439-442.

Morse, S. H., 1994, *Practical Parallel Computing*, Academic Press, New York.

Nagabhushanam, J. and Gaonkar, G. H., 1995, "Automatic Identification of Modal Damping from Floquet Analysis," *Journal of the American Helicopter Society*, Vol. 40, pp. 39-42.

Peters, D. A., 1995, "Fast Floquet Theory and Trim for Multi-Bladed Rotorcraft," Proceedings of the 51st Annual Forum of the American Helicopter Society, Fort Worth, TX, pp. 444-459.

Pfister, G. F., 1995, *In Search of Clusters — The Coming Battle in Lowly Parallel Computing*, Prentice Hall, New Jersey.

Ravichandran, S., Gaonkar, G. H., Nagabhushanam, J. and Reddy, T. S. R., 1990, "A Study of Symbolic Processing and Computational Aspects in Helicopter Dynamics," *Journal of Sound and Vibration*, Vol. 137, pp. 495-507.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D. and Dongarra, J. J., 1996, *MPI: The Complete Reference*, The MIT Press, Cambridge, MA.

Strawn, R. C., 1997, Private Communication.

Subramanian, S., 1996, *Dynamic Stall and Three-Dimensional Wake Effects on Isolated*

Rotor Trim and Stability with Experimental Correlation and Parallel Fast Floquet Analysis, Ph.D. Thesis, College of Engineering, Florida Atlantic University, Boca Raton, FL.

Subramanian, S. and Gaonkar, G. H., 1996, "Parallel Fast-Floquet Analysis of Trim and Stability for Large Helicopter Models," Proceedings of the 22*nd* European Rotorcraft Forum and 13*th* European Helicopter Association Symposium, Brighton, UK, Sep 17-19, pp. 94.1-94.14.

Subramanian, S., Gaonkar, G. H., Nakadi, R. M., and Nagabhushanam, J., 1996, "Parallel Computing Concepts and Methods for Floquet Analysis of Helicopter Trim and Stability," *Journal of the American Helicopter Society*, Vol. 41, pp. 370-382.

Venkataratnam, S., Subramanian, S. and Gaonkar, G. H., 1997, "Fast-Floquet Analysis of Helicopter Trim and Stability With Distributed and Massively Parallel Computing," Proceedings of the 23*rd* European Rotorcraft Forum, Dresden, Germany, Sep 16-18.

Table 1: A Sample of Computational Reliability Parameters
M: Massively Parallel Computing
D: Distributed Computing

System Order, $N + c$	Condition number of the Jacobian matrix for the converged cycle	Eigenvalue condition number for the lag regressive mode	Residual error of the corres-ponding eigenpair
227 (M)	204351.57	2.5092	0.7254E-14
227 (D)	173326.28	2.5181	0.5247E-14
395 (M)	651771.24	2.6393	0.2812E-13
395 (D)	572671.30	2.6412	0.2223E-13

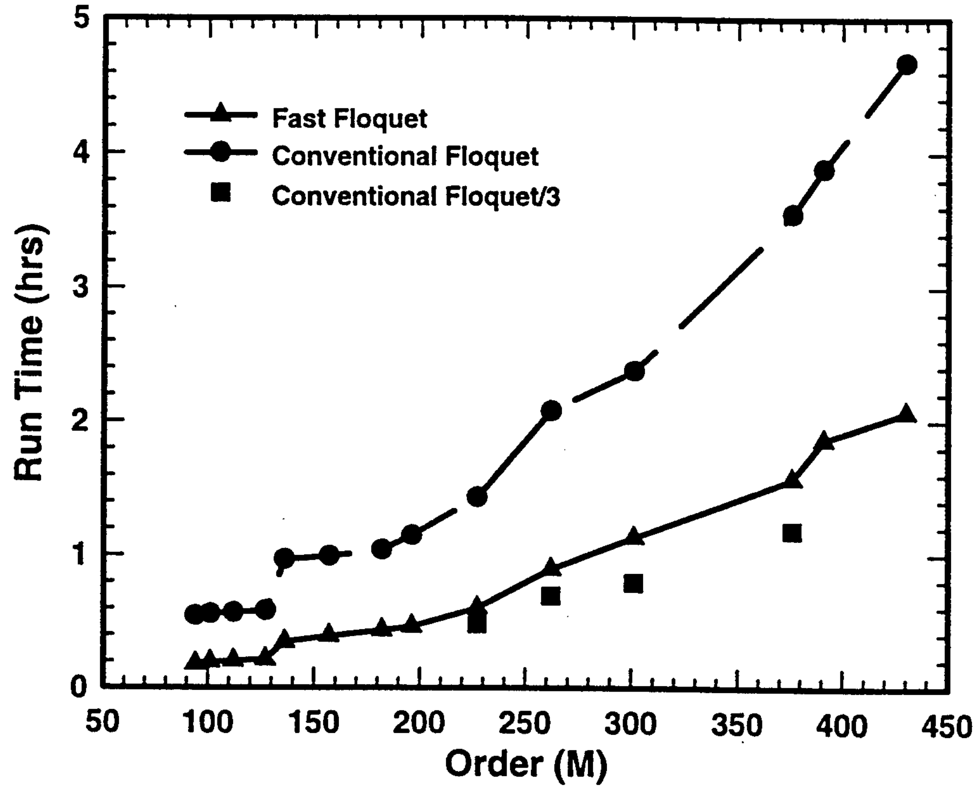

Figure 1: Run-Time Variations of Parallel Analyses Based on Conventional and Fast-Floquet Analyses on a SIMD Computer (MasPar MP-1)

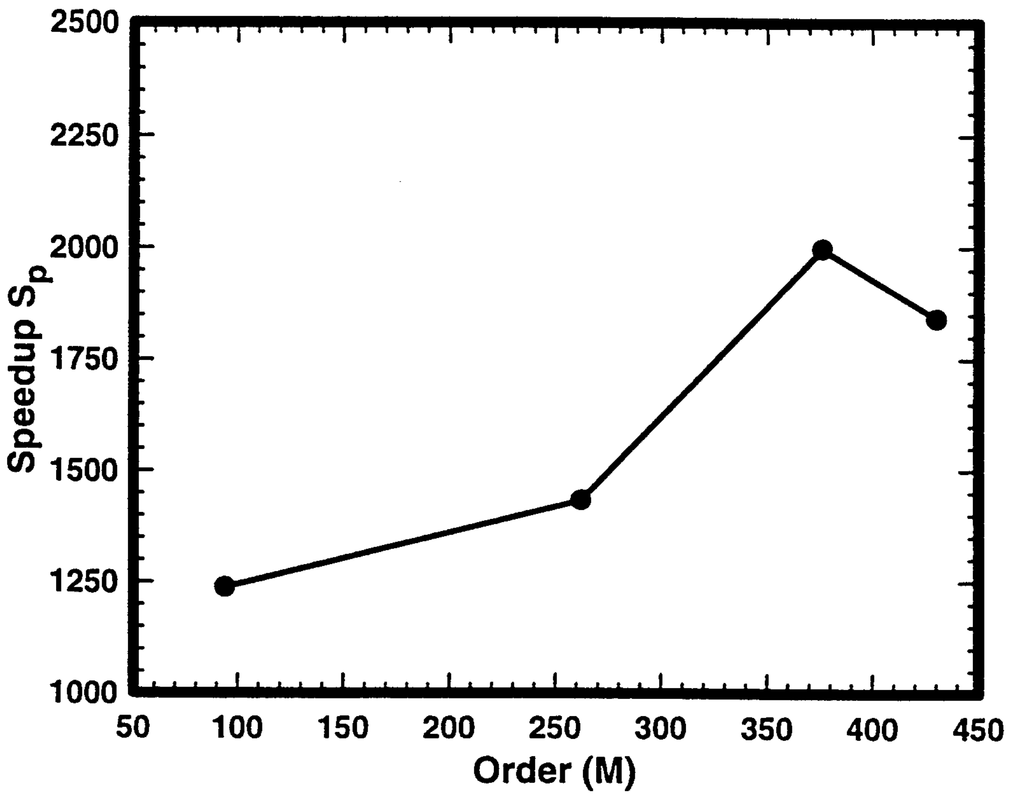

Figure 2: Speedup Versus Model Order for the Parallel Fast-Floquet Analysis on a SIMD Computer (MasPar MP-1)

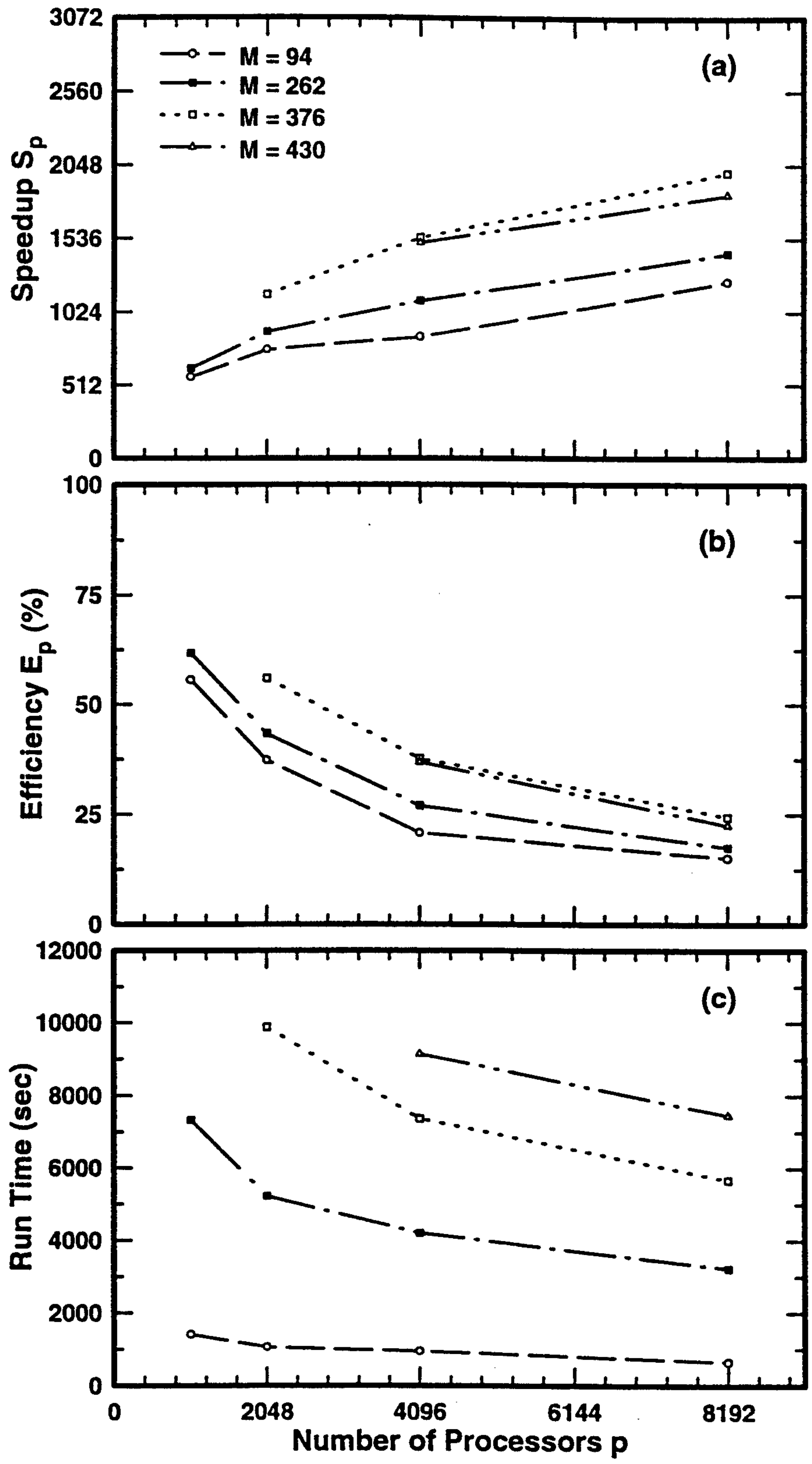

Figure 3: Variations of Speedup, Efficiency and Run Time with the Order and Number of Processors for the Parallel Fast-Floquet Analysis on a SIMD Computer (MasPar MP-1)

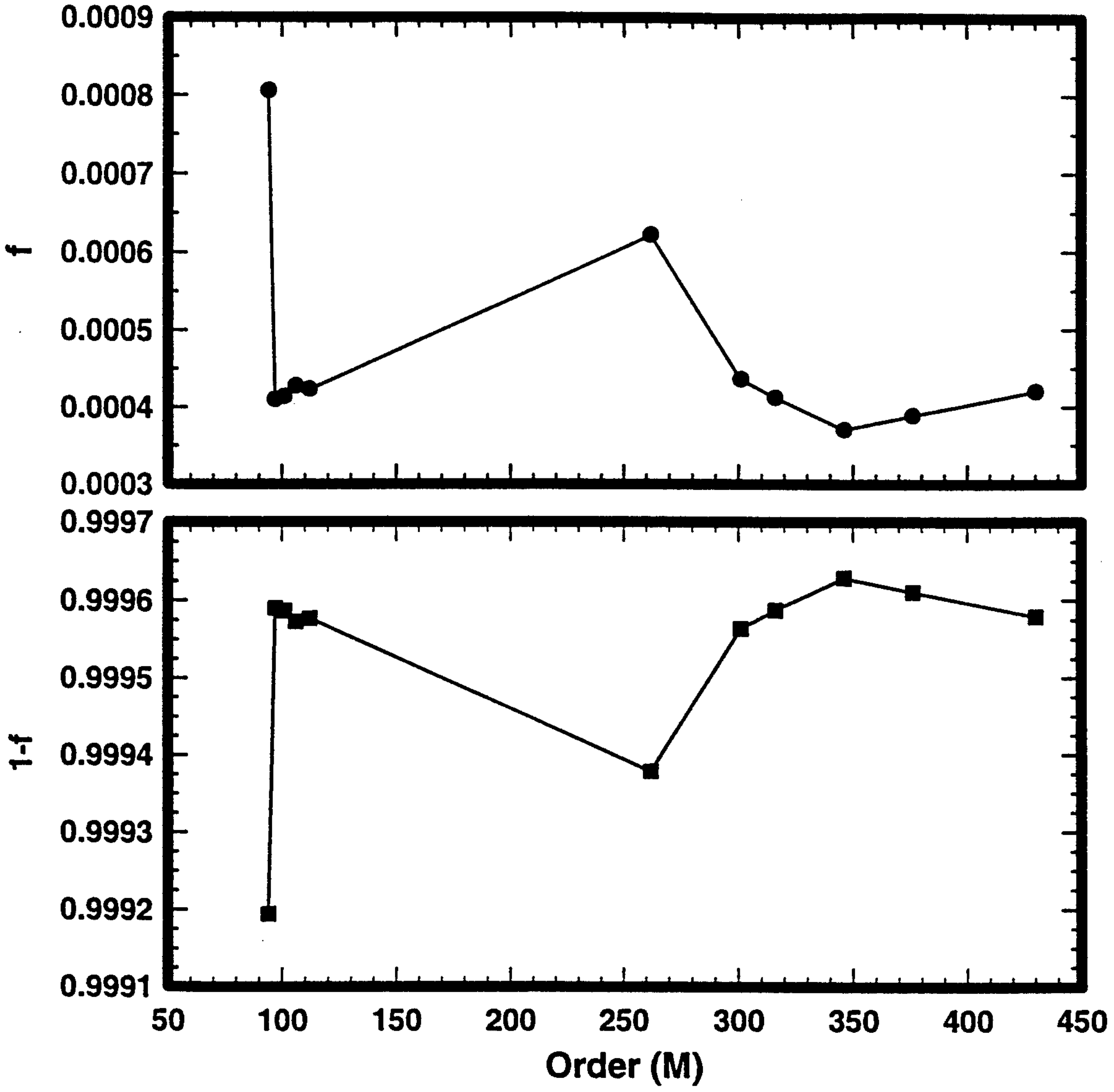

Figure 4: Variations of Sequential and Parallel Fractions with the Order for the Parallel Fast-Floquet Analysis on a SIMD Computer (MasPar MP-1)

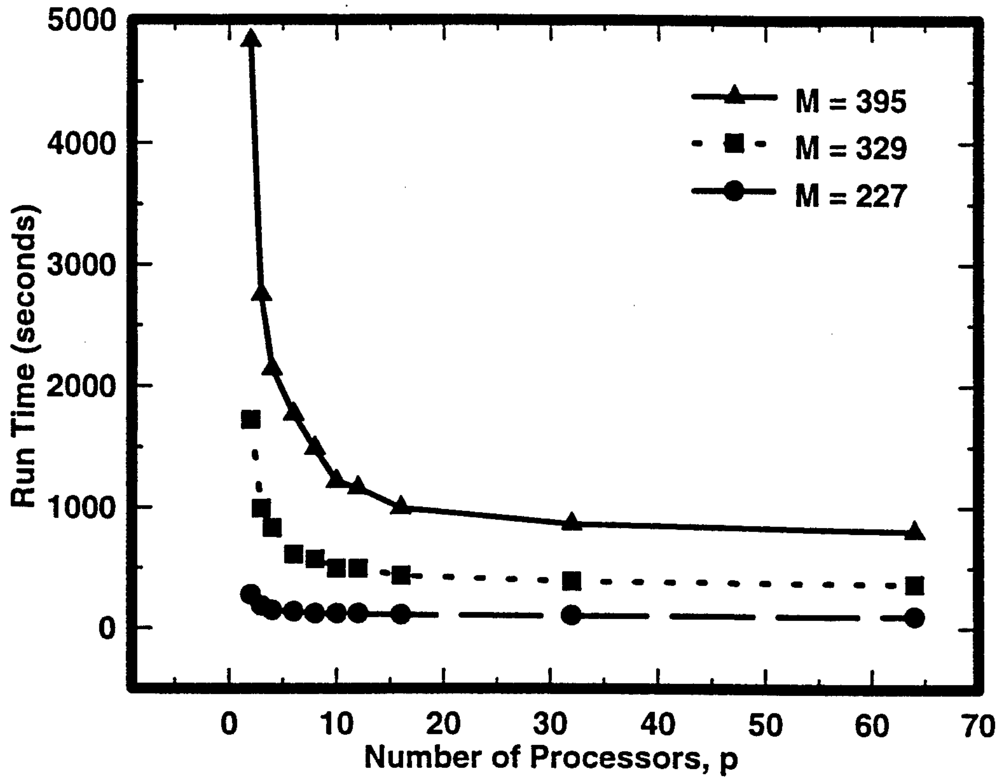

Figure 5: Run-Time Variations with the Order and Number of Processors on a MIMD Computer (IBM SP-2)

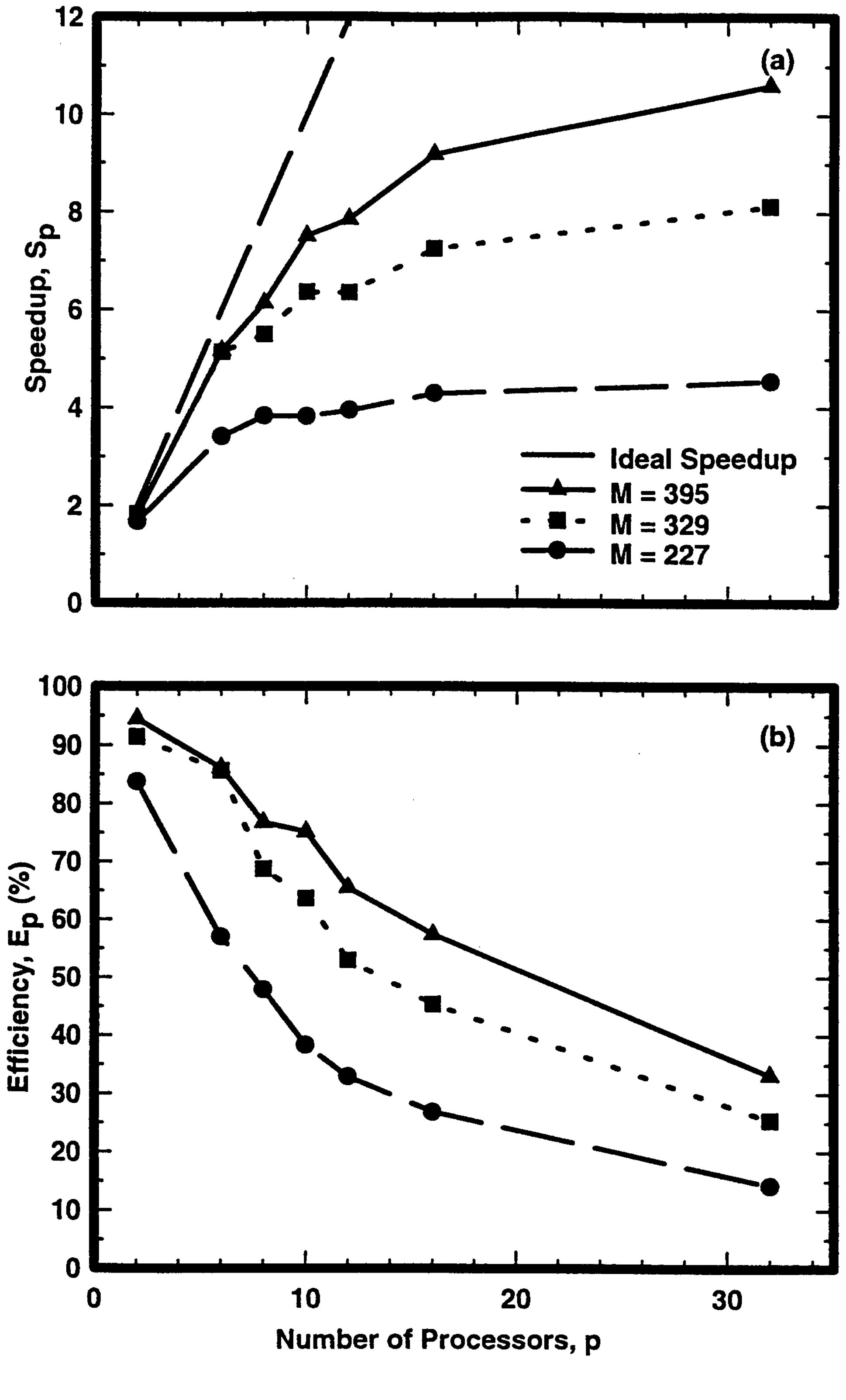

Figure 6: Speedup and Efficiency Variations with the Order and Number of Processors on a MIMD Computer (IBM SP-2)

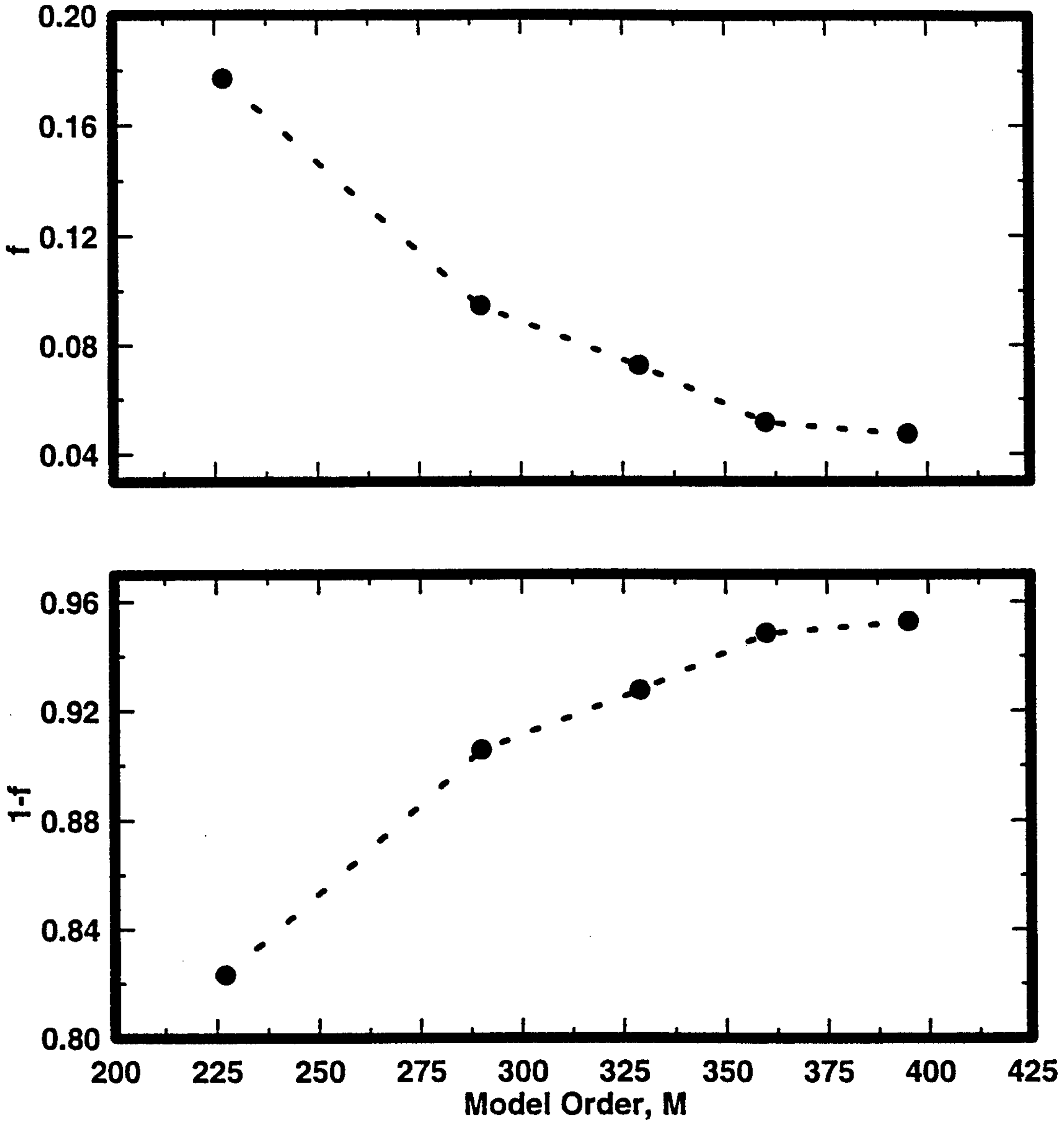

Figure 7: Variations of Serial and Parallel Fractions with the Order on a MIMD Computer (IBM SP-2)

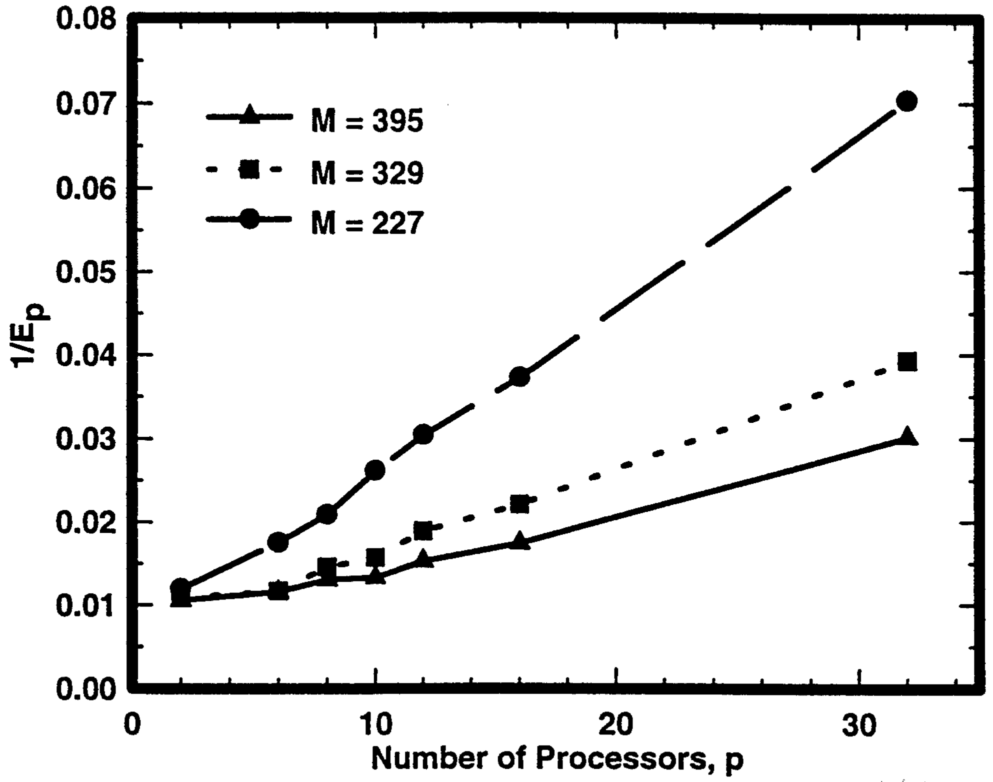

Figure 8: Variations of $1/E_p$ with the Order and Number of Processors on a MIMD Computer (IBM SP-2)

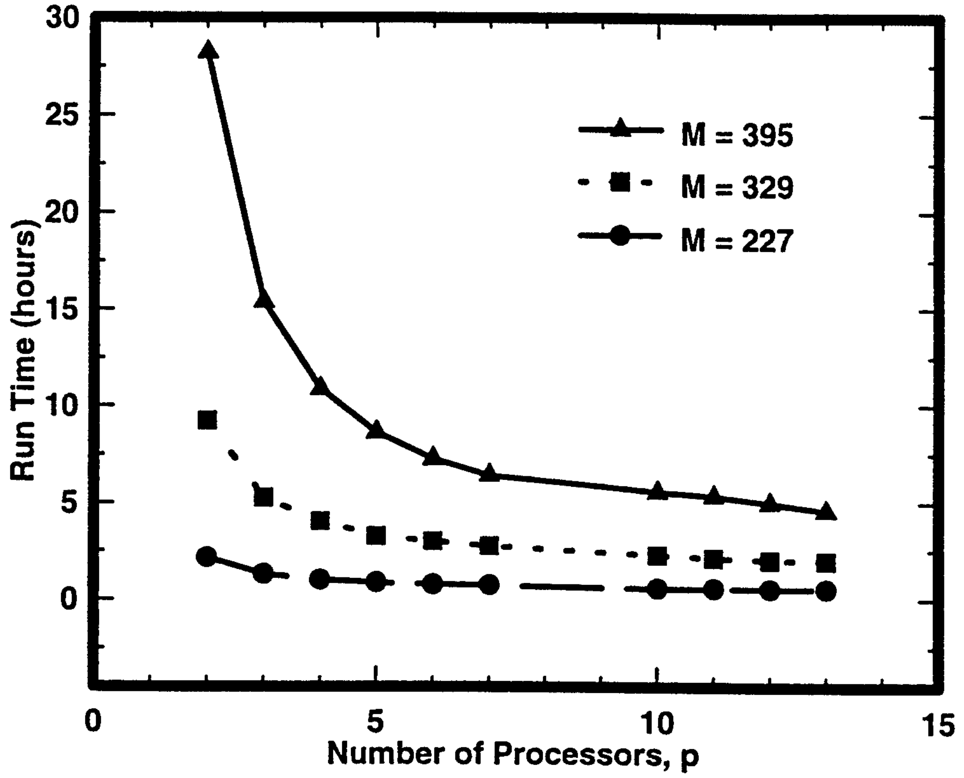

Figure 9: Run-Time Variations with the Order and Number of Processors on a Distributed Computing System

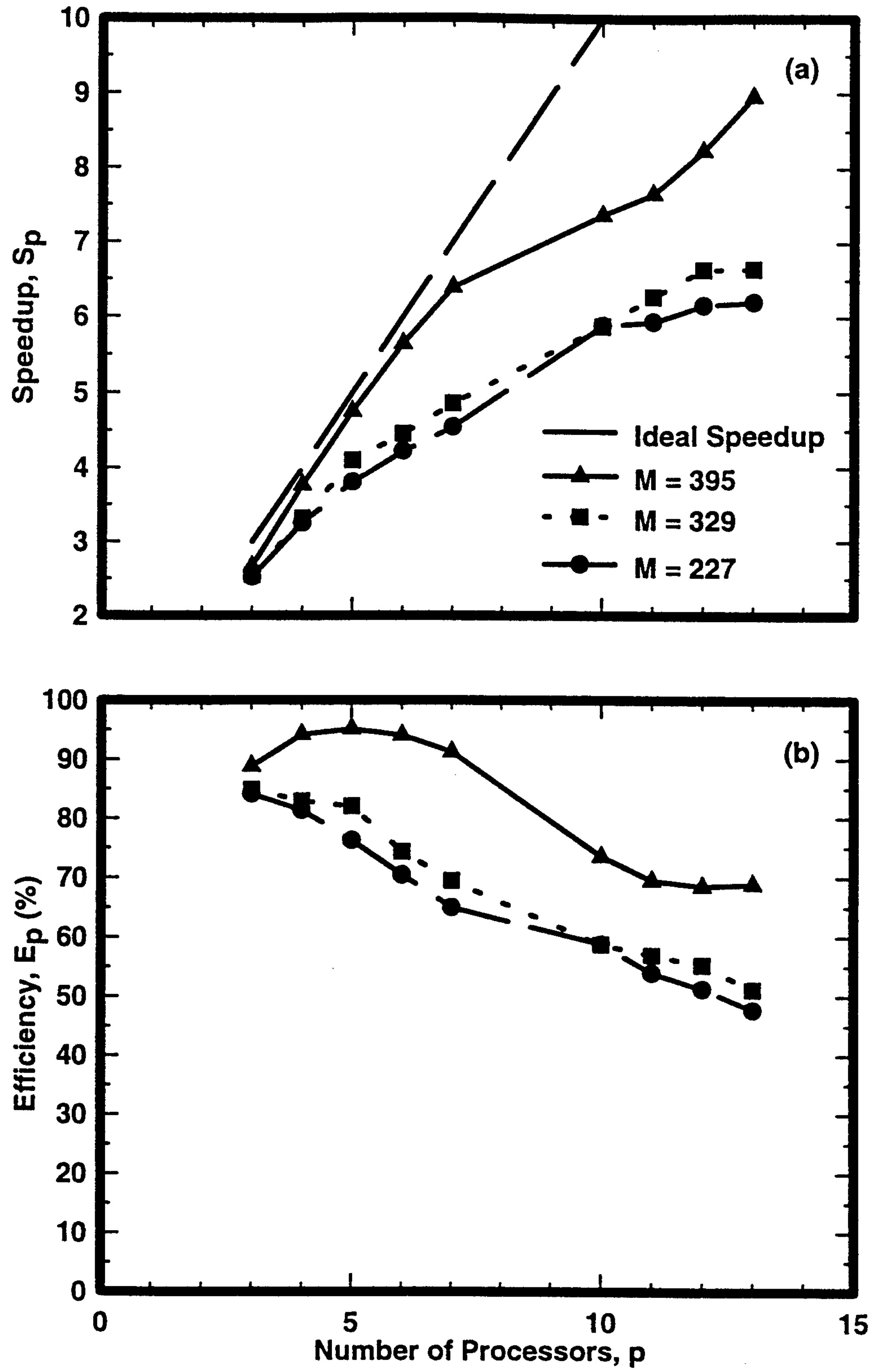

Figure 10: Speedup and Efficiency Variations with the Order and Number of Processors on a Distributed Computing System

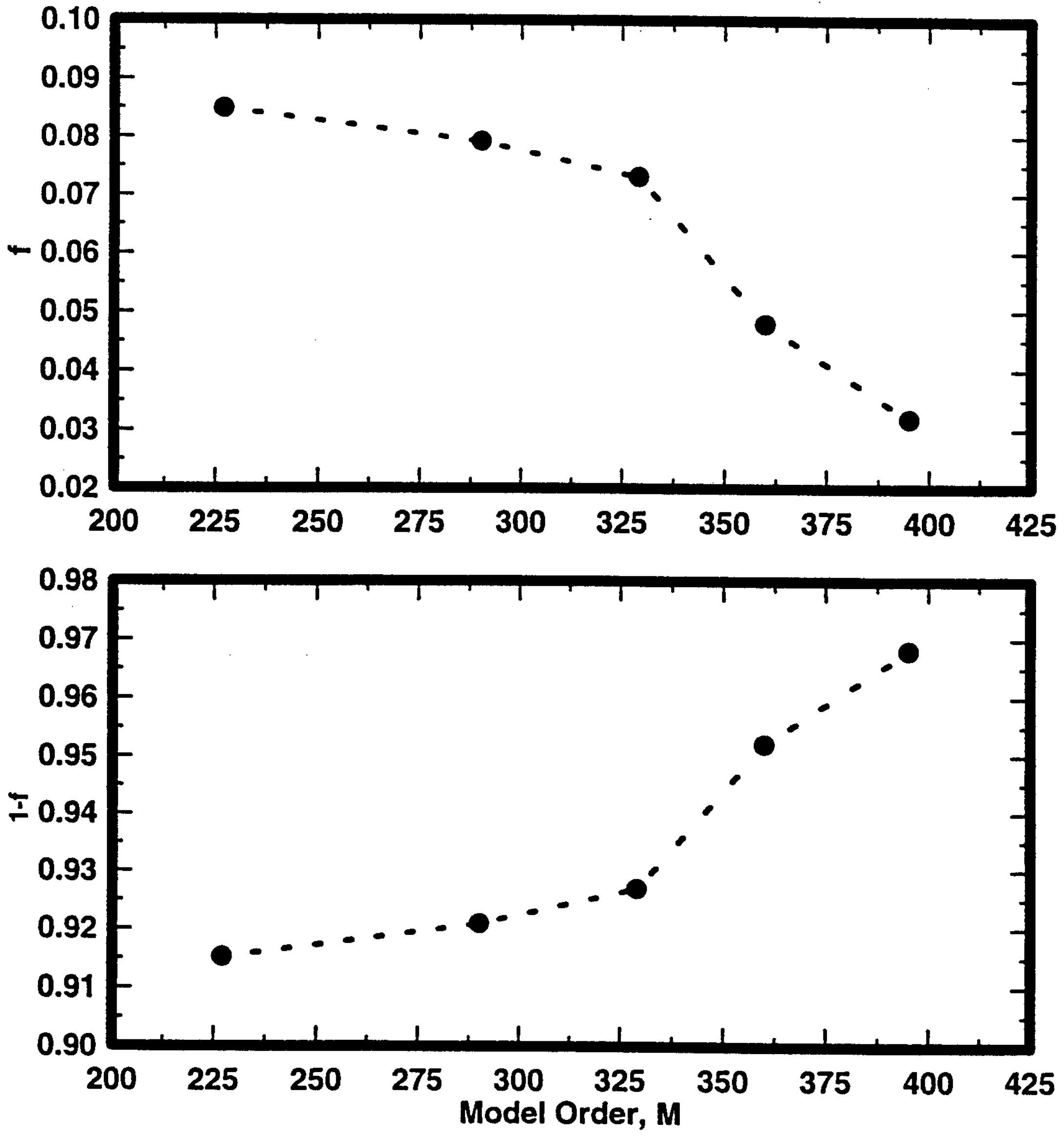

Figure 11: Variations of Serial and Parallel Fractions with the Order on a Distributed Computing System

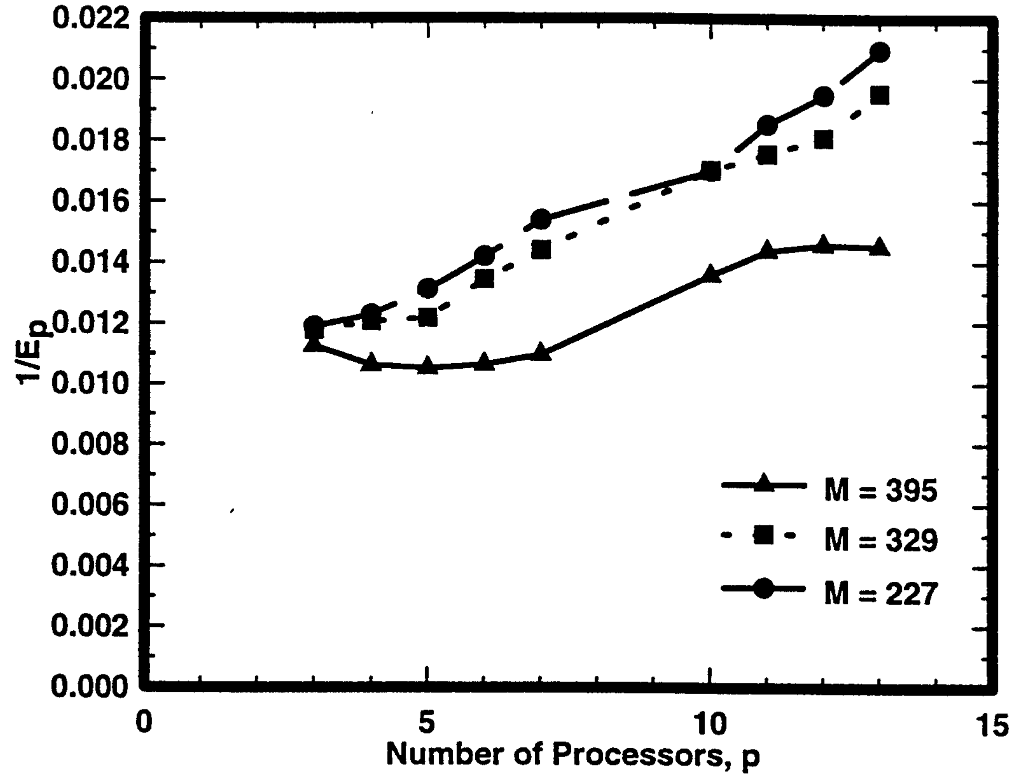

Figure 12: Variations of $1/E_p$ with the Order and Number of Processors on a Distributed Computing System

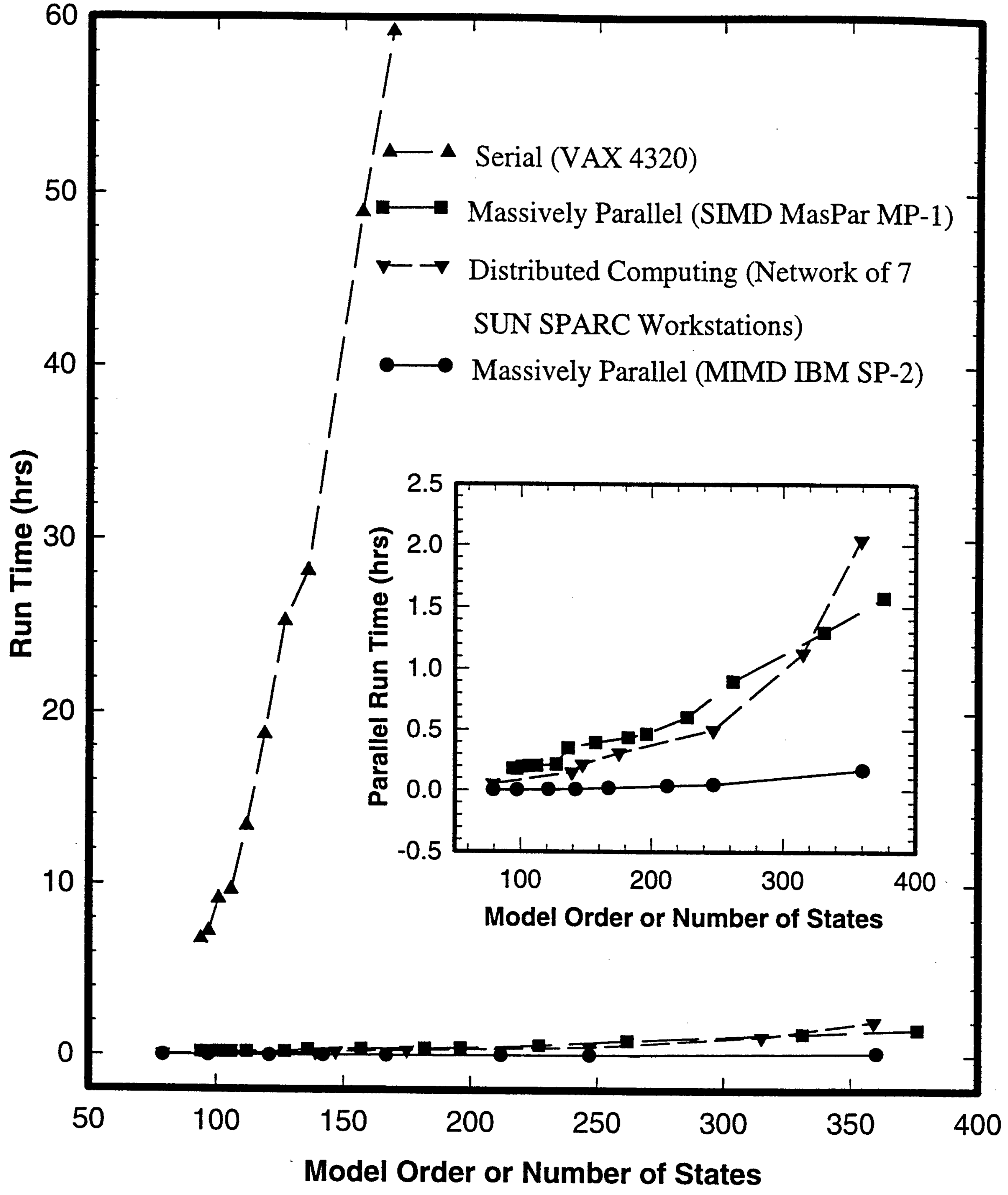

Figure 13: Run-Time Variations with the Order on Serial, Parallel and Distributed Computing Systems

List of Publications

Following is the list of manuscripts submitted or published under ARO sponsorship during the period of this research (including journal references).

1. Nakadi, R. M., "Parallel Computing Concepts and Methods Toward Large-Scale Floquet Analysis of Helicopter Trim and Stability," M.S. Thesis, College of Engineering, Florid Atlantic University, Boca Raton, FL, Aug 1994.

2. Dang, Y. Y., "Turbulence Modeling and Simulation and Related Effects on Helicopter Response with Wake Dynamics Using Finite Elements and Parallelism," M.S. Thesis, College of Engineering, Florid Atlantic University, Boca Raton, FL, Apr 1995.

3. Nagabhushanam, J. and Gaonkar, G. H., "Automatic Identification of Modal Damping from Floquet Analysis," *Journal of the American Helicopter Society,* Vol. 40, (2), 1995.

4. Subramanian, S. and Gaonkar, G. H., "Parallel Fast-Floquet Analysis of Trim and Stability for Large Helicopter Models," Proceedings of the UCLA/ARO Sixth International Workshop on Dynamics and Aeroelastic Modeling of Rotorcraft Systems, Los Angeles, CA, Nov 8-10, 1995.

5. Chunduru, S. J., Subramanian, S. and Gaonkar, G. H., "Dynamic Stall and Wake Effects on Trim and Stability of Hingeless Rotors with Experimental Correlation," Proceedings of the UCLA/ARO Sixth International Workshop on Dynamics and Aeroelastic Modeling of Rotorcraft Systems, Los Angeles, CA, Nov 8-10, 1995.

6. Chunduru, S. J., *Dynamic Stall and Three-Dimensional Wake Effects on Trim, Stability and Loads of Hingeless Rotors with Fast Floquet Theory,* Ph.D. Thesis, College of Engineering, Florid Atlantic University, Boca Raton, FL, Dec 1995.

7. Manjunath, A. R., Chunduru, S. J., Nagabhushanam, J. and Gaonkar, G. H., "Flap-Lag-Torsion Stability in Hover and Forward Flight with Three-Dimensional Wake," *AIAA Journal,* Vol. 34, (1), Jan 1996.

8. Dang, Y. Y., Subramanian, S. and Gaonkar, G. H., "Modeling Turbulence Seen by Multibladed Rotors for Predicting Rotorcraft Response with Three-Dimensional Wake," Proceedings of the AIAA Dynamics Specialists Conference, Salt Lake City, Utah, Apr 18-19, 1996.

9. Dang, Y. Y., Gaonkar, G. H. and Prasad, J. V. R., "Parallel Methods for Turbulence Simulation and Helicopter-Response Prediction," *Journal of the American Helicopter Society,* Vol. 41, (3), Jul 1996.

10. Subramanian, S., *Dynamic Stall and Three-Dimensional Wake Effects on Isolated Rotor Trim and Stability with Experimental Correlation and Parallel Fast-Floquet Analysis,* Ph.D. Thesis, College of Engineering, Florida Atlantic University, Boca Raton, FL, Aug 1996.

11. Subramanian, S. and Gaonkar, G. H., "Parallel Fast-Floquet Analysis of Trim and Stability for Large Helicopter Models," Proceedings of the 22nd European Rotorcraft Forum and 13th European Helicopter Association Symposium, Brighton, U.K., Sep 16-19, 1996.

12. Subramanian, S., Gaonkar, G. H., Nakadi, R. M. and Nagabhushanam, J., "Parallel Computing Concepts and Methods for Floquet Analysis of Helicopter Trim and Stability," *Journal of the American Helicopter Society,* Vol. 41, (4), Oct 1996.

13. Nagabhushanam, J. and Gaonkar, G. H., "Hingeless-Isolated Rotor Aeromechanical Stability in Hover and Forward Flight With Wake Dynamics," Proceedings of the 38th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Kissimme, FL, Apr 7-10, 1997.

14. Subramanian, S., Gaonkar, G. H. and Maier, T. H., "A Theoretical and Experimental Investigation of Hingeless-Rotor Stability and Trim," Proceedings of the 23nd European Rotorcraft Forum Dresden, Germany, Sep 16-18, 1997.

15. Venkataratnam, S., Subramanian, S. and Gaonkar, G. H., "Fast-Floquet Analysis of Helicopter Trim and Stability With Distributed and Massively Parallel Computing,"

Proceedings of the 23nd European Rotorcraft Forum Dresden, Germany, Sep 16-18, 1997.

16. Dang, Y. Y., Subramanian, S. and Gaonkar, G. H., "Modeling Turbulence Seen by Multibladed Rotors for Predicting Rotorcraft Response with Three-Dimensional Wake," *Journal of the American Helicopter Society,* Vol. 42, (4), Oct 1997.

17. Chunduru, S. J., Subramanian, S. and Gaonkar, G. H., "Dynamic Stall and Wake Effects on Trim and Stability of Hingeless Rotors With Experimental Correlation," *Journal of the American Helicopter Society,* Vol. 41, (4), Oct 1997.

18. Subramanian, S., Ma, G. and Gaonkar, G. H., "Lag-Damping Prediction of Hingeless Rotors in Trimmed Flight With Experimental Correlation," Proceedings of the Washington University/ARO Seventh International Workshop on Dynamics and Aeroelastic Stability Modeling of Rotorcraft Systems, St. Louis, MO, Oct 14-16, 1997.

19. Nagabhushanam, J. and Gaonkar, G. H., "Hingeless-Rotor Aeromechanical Stability in Hover and Forward Flight With Wake Dynamics," Proceedings of the Washington University/ARO Seventh International Workshop on Dynamics and Aeroelastic Stability Modeling of Rotorcraft Systems, St. Louis, MO, Oct 14-16, 1997.

20. Venkataratnam, S., Subramanian, S. and Gaonkar, G. H., "Fast-Floquet Analysis of Helicopter Trim and Stability With Distributed and Massively Parallel Computing," Proceedings of the Washington University/ARO Seventh International Workshop on Dynamics and Aeroelastic Stability Modeling of Rotorcraft Systems, St. Louis, MO, Oct 14-16, 1997.

21. Subramanian, S., Venkataratnam, S. and Gaonkar, G. H., "Serial and Parallel Fast-Floquet Analyses in Rotorcraft Aeroelasticity," Proceedings of the 4th International Symposium on Fluid-Structure Interaction, Aeroelasticity, Flow-Induced Vibration and Noise, International Mechanical Engineering Congress and Exposition, Dallas, TX, Nov 16-21, 1997.

22. Gaonkar, G. H., Venkataratnam, S. and Subramanian, S., "Parallel Fast-Floquet Analysis for Helicopter Trim and Stability," MHPCC Application Briefs, Maui High Performance Computing Center, Maui, HI, 1997.

List of Personnel Supported

The estimated level of effort for this contract is about 65 man-months including the efforts of the principal investigator, Prof. Gopal H. Gaonkar. The personnel supported by this project are as follows:

1. Mr. R. M. Nakadi, M. S. degree completed in Fall 1994* (part of Summer 1994).
2. Mr. Y. Y. Dang, M.S. degree completed in Spring 1995* (Summer 94, part of Fall 1994 and Spring 1995).
3. Dr. S. J. Chunduru, Ph.D. degree completed in Fall 1995*.
4. Dr. S. Subramanian, Ph.D. degree completed in Summer 1996*.
5. Dr. S. Subramanian, Post Doctoral Fellow, Aug 1996-Oct 1997.
6. Mr. G. Ma, Ph.D. student, since Aug 1994*.
7. Mr. S. Venkataratnam, Ph.D. student, since Aug 1994*.
8. Mr. Z. Zhang, Ph.D. student, since Aug 1994* (discontinued).

*partially supported by the Department of Mechanical Engineering, Florida Atlantic University.